#119 SEPTEMBER 1986

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

SMOOTH ALGORITHMS

MS-DOS Directory Traversal

Turbo Boards Review

Radix Sort

Does Turbo Prolog Measure Up?

Crawling Memory Test

0 38351 16562

USE THE BRAINS YOUR IBM WASN'T BORN WITH.

Right at your fingertips in CompuServe's IBM® Forums.

In the **IBM New Users Forum** you'll swap ideas with other new PC users, learn to use Forum features, and pose even basic questions to PC experts.

Our **IBM Junior Forum** gives PCjr® users a reliable source for tips on software, hardware, telecommunications, games and other interests.

In the **IBM Software Forum** you'll trade tips with other IBM PC and AT users on utility software, word processing, DOS and other operating systems.

Visit the **IBM Communications Forum** for advice on the features and compatibility of communications software and hardware, PC Bulletin Boards, micro-mainframe interfaces and more.

The **IBM Hardware Forum** addresses hardware topics of all types, plus product updates and announcements.

Easy access to free software.

- *Download first-rate, non-commercial, user-supported software and utility programs.*
- *Take advantage of CompuServe's inexpensive weeknight and weekend rates (when Forums are most active, and standard online charges are just 10¢ a minute).*
- *Go online in most major metropolitan areas with a local phone call.*
- *And receive a **$25.00 Introductory Usage Credit** with purchase of your CompuServe Subscription Kit.*

Information you simply can't find anywhere else.

Use the Forum *Message Board* to send and receive electronic messages, and pose specific questions to other IBM and compatible owners.

Join ongoing, real-time discussions in a Forum *Conference.*

Search our unparalleled Forum *Data Libraries* for free software, user tips, transcripts of online conferences and more.

Enjoy other useful services like:

- ***Popular Computer Magazines**—electronic editions, for your reading pleasure. Including Dr. Dobb's Journal and Computer Language.*
- ***Other CompuServe Forums**—supporting LOTUS® products like Symphony™ and 1-2-3.™ Borland International,® Ashton-Tate,® Digital Research,® MicroPro,® Microsoft® and other software. Also Pascal, Basic, C, Forth, Assembly and other programming languages.*

All you need is your IBM or IBM-compatible computer and a modem ...or almost any other computer.

To buy your Subscription Kit, see your nearest computer dealer. Suggested retail price is $39.95. To receive our free brochure, or to order direct, call 800-848-8199 (in Ohio, call 614-457-0802). If you're already a CompuServe subscriber, type GO IBMNET (the IBM Users Network) at any ! prompt to see what you've been missing.

CompuServe®

Information Services, P.O. Box 20212
5000 Arlington Centre Blvd., Columbus, Ohio 43220

800-848-8199

In Ohio, Call 614-457-0802
An H&R Block Company

Circle **no. 237** on reader service card.

Those who insist on C compiler performance are very big on Mark Williams.

And the compiler is just part of our total C Programming System.

NEW 3.0 ENHANCEMENTS

These and other powerful utilities now *included* in the C Programming System:

- make: compiles only what's necessary from multiple modules, a powerful programming discipline
- diff: identifies differences between two files
- m4: macroprocessor expression editing and substitution
- egrep: extended pattern search
- MicroEMACS: full screen editor with source

COMPILER FEATURES

- Runs under MS-DOS
- Full Kernighan & Ritchie C with recent extensions including void and enum
- Register variables for fast, compact code
- Full UNIX™ compatibility and complete libraries
- 8087 Support
- One-step compiling
- English error messages
- ROMable code
- Large and small memory models
- MS-DOS linker compatibility
- Linker, assembler, archiver
- Extensive third party library support

csd C SOURCE DEBUGGER

- Debugs at C source level without assembly language
- Separate evaluation, source, program and history windows
- Can execute any C expression
- Capabilities of a C interpreter, but runs in real time
- Set trace points on any statement or variable

New! C for the Atari ST. $179.95 Call for features.

Mark Williams' C compiler has earned a place in some very big companies for some very good reasons: it proves the benchmarks right with the speed, code density, consistent performance and expert support required in professional development environments.

But a total development tool shouldn't stop with compiling. Or go on and on with extras that add up and up.

Only Mark Williams' C Programming Systems *includes* the csd C Source Debugger with true source level debugging to speed your programming job.

And only Mark Williams' new 3.0 version *includes* utilities like "make" to make quick work of even the largest projects.

From source code to final product, only one takes you all the way: Mark Williams' C Programming System. All for only $495. **Ask about our 60-day money back guarantee when you call 1-800-692-1700 to order today.*** You'll be big on the total C Programming system from Mark Williams, too.

*In Illinois call 312-472-6659.

Mark Williams Company
1430 West Wrightwood
Chicago, Illinois 60614

Instant Replay

NEW
Synthetic Intelligence
Patent Pending

Generates:
DEMOS, TUTORIALS,
PROTOTYPES, PRESENTATIONS,
TIMED KEYBOARD MACROS, AND
MENU SYSTEMS

Instant Replay memorizes how you ran your program and Instantly generates a Demo Replay. Keystrokes and pause times, inserted prompts, pop-ups and prototypes are all memorized.

Screen Painter for creating 100% user designable pop-ups and Menu Windows. **"Excellent, intuitive, easiest-to-use screen generator reviewed" PC Magazine .**

Run-Time Screen Scanner and File Scanner for creating Menu systems and presentations.

Screen Grabber so you can edit and include any screen.

Text Editor - fast, full function, with pull-down menus that can be tailored. Memorize it and distribute it as a Demo!

200 Page Manual, 4 diskettes, 60 Day money back guarantee.
(Not Copy Protected)

Call or Write. We accept Visa, Amex, Master Card, COD, PO.
Dealer Inquiries Welcome.

____	**Instant Replay**(tm)	$ 89.95
____	**Demo Diskette**	$ 5.00
____	**Dealer Poster**	$ 3.00

Nostradamus

Nostradamus Inc. / 5320 South 900 East, Suite 110
SLC, Utah 84117 / Order by phone (801) 261-0769

For IBM PC / XT / AT and True Compatibles

Ask about Programming Libraries

Circle **no. 251** on reader service card.

SEPTEMBER 1986

CONTENTS

VOLUME 11, ISSUE 9

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

ARTICLES

Algorithms for smooth curves ▶

Radix sort ▶

REVIEWS

Is it really PROLOG? ▶

Turbo boards for the IBM PC ▶

COLUMNS

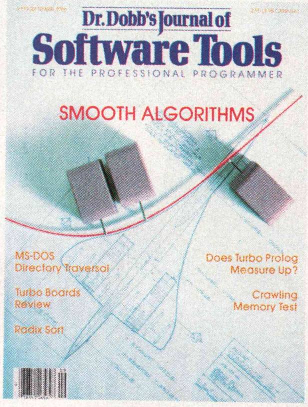
Directory trees ▶

Memory test with the 68000 ▶

FORUM

A growing sanity ▶

PROGRAMMER'S SERVICES



About the Cover
The architectural illustration for this cover was done by Robert H. Frank. The photograph was done by Michael Carr/Pacific Horizons.

This Issue
Many algorithms now used on computers have their roots in mechanical processes. This month, we present two such algorithms; one that replaces the draftsman's spline and one that sorts in an old fashioned (but sometimes faster) way. Michael Swaine offers some thoughts on Borland's latest product, and we present a review of several PC speed boosters. Allen Holub and Ray Duncan provide tasty tidbits, and Jan Steinman brings us a very unusual memory test.

Next Issue
In October, we'll take a close look at Intel's 80386 chip. What does it really offer, and how do you upgrade from the 80286? We'll also demonstrate some interesting code for the NS320xx chip set.

LANGUAGES THAT ARE CAUSING THE BIGGEST PROGRAMMING BACKLOG IN HISTORY ARE ALSO EATING NICE BIG HOLES IN OUR POCKETS.

Whether it's BASIC, COBOL, Pascal, "C", or a data base manager, you're being held back.

Held back because the language has frustrating limitations, and the programming environment isn't intuitive enough to keep track of what you're working on.

In the real world, there's pressure to do more impressive work, in less time, and for more clients.

We've been given some incredibly powerful hardware in recent times, but the languages aren't a whole lot better than they were 20 years ago.

So, whatever language you have chosen, by now you feel it's out to get you — because it is.

Sure, no language is perfect, but you have to wonder, "Am I getting all I deserve?"

And, like money, you'll never have enough.

Pretty dismal, huh?

We thought so, too.

So we did something about it.

We call it CLARION™.

You'll call it *"incredible."*

Distributed on 7 diskettes, CLARION consists of over 200,000 lines of code, taking 3+ years to hone to "world-class" performance.

With CLARION you can write, compile, run and debug complex applications in a New York afternoon.

Even if you're in Savannah.

It gives you the power and speed to create screens, windows and reports of such richness and clarity you would never attempt them with any other language.

Because *you* would have to write the code.

With CLARION you simply design the screens using our SCREENER utility and then CLARION writes the source code AND compiles it for you in seconds.

Likewise, you can use REPORTER to create reports.

Remember, only CLARION can recompile and display a screen or report layout for modification.

And with no time wasted.

All the power and facilities you need to write great programs, faster than you ever dreamed of.

Programs that are easy to use.

Programs that are a pleasure to write.

And to you that means true satisfaction.

You've coveted those nifty pop-up help windows some major applications feature. But you can't afford the time and energy it takes to write them into your programs.

That's the way it used to be.

So we fixed that, too.

CLARION's HELPER is an interactive utility that let's you design the most effective pop-up help screens that you can imagine. And they're "context sensitive," meaning you can have help for every field in your application.

Unlike the other micro languages, CLARION provides declarations, procedures, and functions to process dates, strings, screens, reports, indexed files, DOS files and memory tables.

Imagine making source program changes with the CLARION EDITOR. A single keystroke terminates the EDITOR, loads the COMPILER, compiles the program, loads the PROCESSOR and executes the program. It's that easy!

Our data management capabilities are phenomenal. CLARION files permit any number of composite keys which are updated dynamically.

A file may have as many keys as it needs. Each key may be composed of any fields in any order. And key files are updated whenever the value of the key changes.

Like SCREENER and REPORTER, CLARION's FILER utility also has a piece of the CLARION COMPILER. To create a new file, you name the Source Module. Then you name the Statement Label of a file structure within it.

FILER will also automatically rebuild existing files to match a changed file structure. It creates a new record for every existing record, copying the existing fields and initializing new ones.

Sounds pretty complicated, huh? Not with CLARION's documentation and on-line help screens. If you are currently competent in BASIC, Pascal or "C" you can be writing CLARION applications in a day. In two days you won't believe the eloquence of your CLARION programs.

Okay, now for the best part of all. You can say it in CLARION for $295.00—plus shipping and handling. All you need is an IBM® PC, XT, AT or true compatible, with 320 KB of memory, a hard disk drive, and a parallel port.

And we'll allow a full 30 day evaluation period. If you're not satisfied with CLARION, simply return it in its original condition for a full refund.

If you're not quite ready to take advantage of this no-risk opportunity, ask for our detailed 16 page color brochure. It vividly illustrates the elegance of CLARION. Consider it a preview of programming in the fast lane.

Either way, the 800 call's a freebie.

SAY IT IN

CLARION™

Dept. A2ST/4

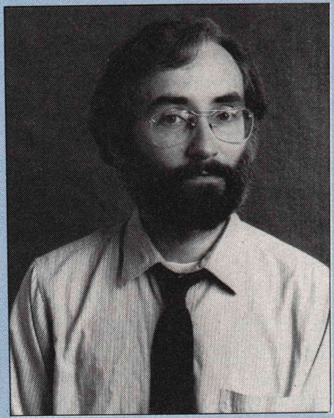
1-800-354-5444

VISA | AMERICAN EXPRESS | MasterCard

BARRINGTON SYSTEMS, INC. 150 EAST SAMPLE ROAD POMPANO BEACH, FLORIDA 33064 305/785-4555

Circle **no. 115** on reader service card.

EDITORIAL



Sometimes the Next Big Thing is a neglected old thing.

When I was a graduate student I took all the AI classes and seminars I could. In a class taught by Gene Freuder, who had just arrived from MIT where he had studied under Patrick Henry Winston, I wrote an expert system for draw poker. At that time, expert systems interested me, but not as much as the challenge, posed by another of my professors, Doug Hofstadter, of writing a program that could generate analogies, a problem in the area of machine learning. My expert system worked, after a fashion, and I never got anywhere with the machine learning program. Looking back over the past few years, it may seem that the same could be said for the entire field of AI: expert systems work; machine learning never got started.

I once believed that, but now I think I was wrong. Research in machine learning has certainly taken a back seat to expert systems work since my graduate school days. The results are evident: By focusing on narrow problem areas and domain-specific knowledge, workers in expert systems have created some very powerful practical tools and turned a technique into an industry. The Business section of the July 7, 1986, San Jose, California, *San Jose Mercury News*, amid grumblings about the industrywide slump in software, computers, and electronics, noted that one expert systems company, Teknowledge, was doing spectacularly well.

Machine learning never makes it into the *Mercury News*; nevertheless, research in machine learning has continued quietly over the years—and not without results. There are, for example, programs whose performance improves with experience and programs that develop new problem-solving heuristics. There are programs that produce interesting analogies.

I hope that machine learning is about to capture the imaginations of programmers and venture capitalists. The potential benefits from work in machine learning are, I believe, far greater than the benefits of expert systems work. Understanding natural language, for example, requires the ability to learn new concepts rather than requiring a body of expert knowledge. And by the very nature of the learning process, it seems that any learning program would have to be general in application, suggesting that we would only need one. Given one, we could clone it.

Also by the nature of the learning process, the performance of learning programs will probably be unreliable, and this may force us to start looking at programs in a different way. Induction and generalization seem incompatible with the provability and reliability necessary for Star Wars-type projects. Making mistakes may well be an inextricable component of learning. We may one day find ourselves praising programs for their efforts as much as for their results.

The review of PC speed-up boards in this issue was completed before we knew of this manufacturer of a comparable board:
STD (Seattle Telecom & Data Inc.)
2637 151st Pl. NE
Redmond, WA 98052
(206) 883-8440

Michael Swaine

Michael Swaine
editor-in-chief

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing, Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

Article Submissions: Send manuscripts and disk (with article and listings) to the Assistant Editor.

Address Correction Requested: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0888-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; within CA (619) 485-9623 or 566-6947. For order problems call (415) 366-3600.

Subscription Rates: $29.97 per year, U.S. Foreign rates: $63.97, air; $51.97, surface. Foreign subscriptions must be prepaid in U.S. dollars, drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service, Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX: 620430 (WUI).

People's Computer Company
Dr. Dobb's Journal of Software Tools is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

AZTEC — The C for Microcomputers

PC-DOS, MS-DOS, CP/M-86, Macintosh, Amiga, Apple II, CP/M-80, Radio Shack, Commodore, XENIX, ROM, and Cross Development systems

MS-DOS, PC-DOS, CP/M-86, XENIX, 8086/80x86 ROM

Manx Aztec C86

"A compiler that has many strengths . . . quite valuable for serious work"

Computer Language review, February 1985

Great Code: Manx Aztec C86 generates fast executing compact code. The benchmark results below are from a study conducted by Manx. The Dhrystone benchmark (CACM 10/84 27:10 p1018) measures performance for a systems software instruction mix. The results are without register variables. With register variables, Manx, Microsoft, and Mark Williams run proportionately faster, Lattice and Computer Innovations show no improvement.

	Execution Time	Code Size	Compile/ Link Time
Dhrystone Benchmark			
Manx Aztec C86 3.3	34 secs	5,760	93 secs
Microsoft C 3.0	34 secs	7,146	119 secs
Optimized C86 2.20J	53 secs	11,009	172 secs
Mark Williams 2.0	56 secs	12,980	113 secs
Lattice 2.14	89 secs	20,404	117 secs

Great Features: Manx Aztec C86 is bundled with a powerful array of well documented productivity tools, library routines and features.

Optimized C compiler	Symbolic Debugger
AS86 Macro Assembler	LN86 Overlay Linker
80186/80286 Support	Librarian
8087/80287 Sensing Lib	Profiler
Extensive UNIX Library	DOS, Screen, & Graphics Lib
Large Memory Model	Intel Object Option
Z (vi) Source Editor -c	CP/M-86 Library -c
ROM Support Package -c	INTEL HEX Utility -c
Library Source Code -c	Mixed memory models -c
MAKE, DIFF, and GREP -c	Source Debugger -c
One year of updates -c	CP/M-86 Library -c

Manx offers two commercial development systems, Aztec C86-c and Aztec C86-d. Items marked -c are special features of the Aztec C86-c system.

Aztec C86-c Commercial System	**$499**
Aztec C86-d Developer's System	**$299**
Aztec C86-p Personal System	**$199**
Aztec C86-a Apprentice System	**$49**

All systems are upgradable by paying the difference in price plus $10.

Third Party Software: There are a number of high quality support packages for Manx Aztec C86 for screen management, graphics, database management, and software development.

C-tree $395	**Greenleaf $185**
PHACT $250	**PC-lint $98**
HALO $250	**Amber Windows $59**
PRE-C $395	**Windows for C $195**
WindScreen $149	**FirsTime $295**
SunScreen $99	**C Util Lib $185**
PANEL $295	**Plink-86 $395**

MACINTOSH, AMIGA, XENIX, CP/M-68K, 68k ROM

Manx Aztec C68k

"Library handling is very flexible . . . documentation is excellent . . . the shell a pleasure to work in . . . blows away the competition for pure compile speed . . . an excellent effort."

Computer Language review, April 1985

Aztec C68k is the most widely used commercial C compiler for the Macintosh. Its quality, performance, and completeness place Manx Aztec C68k in a position beyond comparison. It is available in several upgradable versions.

Optimized C	Creates Clickable Applications
Macro Assembler	Mouse Enhanced SHELL
Overlay Linker	Easy Access to Mac Toolbox
Resource Compiler	UNIX Library Functions
Debuggers	Terminal Emulator (Source)
Librarian	Clear Detailed Documentation
Source Editor	C-Stuff Library
MacRam Disk -c	UniTools (vi,make,diff,grep) -c
Library Source -c	One Year of Updates -c

Items marked -c are available only in the Manx Aztec C86-c system. Other features are in both the Aztec C86-d and Aztec C86-c systems.

Aztec C68k-c Commercial System	**$499**
Aztec C68d-d Developer's System	**$299**
Aztec C68k-p Personal System	**$199**
C-tree database (source)	**$399**
AMIGA, CP/M-68k, 68k UNIX	**call**

Apple II, Commodore, 65xx, 65C02 ROM

Manx Aztec C65

"The AZTEC C system is one of the finest software packages I have seen"

NIBBLE review, July 1984

A vast amount of business, consumer, and educational software is implemented in Manx Aztec C65. The quality and comprehensiveness of this system is competitive with 16 bit C systems. The system includes a full optimized C compiler, 6502 assembler, linkage editor, UNIX library, screen and graphics libraries, shell, and much more. The Apple II version runs under DOS 3.3, and ProDOS, Cross versions are available.

The Aztec C65-c/128 Commodore system runs under the C128 CP/M environment and generates programs for the C64, C128, and CP/M environments. Call for prices and availability of Apprentice, Personal and Developer versions for the Commodore 64 and 128 machines.

Aztec C65-c ProDOS & DOS 3.3	**$399**
Aztec C65-d Apple DOS 3.3	**$199**
Aztec C65-p Apple Personal system	**$99**
Aztec C65-a for learning C	**$49**
Aztec C65-c/128 C64, C128, CP/M	**$399**

Distribution of Manx Aztec C

In the USA, Manx Software Systems is the sole and exclusive distributor of Aztec C. Any telephone or mail order sales other than through Manx are unauthorized.

Manx Cross Development Systems

Cross developed programs are edited, compiled, assembled, and linked on one machine (the HOST) and transferred to another machine (the TARGET) for execution. This method is useful where the target machine is slower or more limited than the HOST, Manx cross compilers are used heavily to develop software for business, consumer, scientific, industrial, research, and educational applications.

HOSTS: VAX UNIX ($3000), PDP-11 UNIX ($2000), MS-DOS ($750), CP/M ($750), MACINTOSH ($750), CP/M-68k ($750), XENIX ($750).

TARGETS: MS-DOS, CP/M-86, Macintosh, CP/M-68k, CP/M-80, TRS-80 3 & 4, Apple II, Commodore C64, 8086/80x86 ROM, 68xxx ROM, 8080/8085/Z80 ROM, 65xx ROM.

The first TARGET is included in the price of the HOST system. Additional TARGETS are $300 to $500 (non VAX) or $1000 (VAX).

Call Manx for information on cross development to the 68000, 65816, Amiga, C128, CP/M-68K, VRTX, and others.

CP/M, Radio Shack, 8080/8085/Z80 ROM

Manx Aztec CII

"I've had a lot of experience with different C compilers, but the Aztec C80 Compiler and Professional Development System is the best I've seen."

80-Micro, December, 1984, John B. Harrell III

Aztec C II-c (CP/M & ROM)	**$349**
Aztec C II-d (CP/M)	**$199**
C-tree database (source)	**$399**
Aztec C80-c (TRS-80 3 & 4)	**$299**
Aztec C80-d (TRS-80 3 & 4)	**$199**

How To Become an Aztec C User

To become an Aztec C user call 1-800-221-0440 or call 1-800-832-9273 (800-TEC WARE). In NJ or outside the USA call 201-530-7997. Orders can also be telexed to 4995812.

Payment can be by check, COD, American Express, VISA, Master Card, or Net 30 to qualified customers.

Orders can also be mailed to Manx Software Systems, Box 55, Shrewsbury, NJ 07701.

How To Get More Information

To get more information on Manx Aztec C and related products, call 1-800-221-0440, or 201-530-7997, or write to Manx Software Systems.

30 Day Guarantee

Any Manx Aztec C development system can be returned within 30 days for a refund if it fails to meet your needs. The only restrictions are that the original purchase must be directly from Manx, shipped within the USA, and the package must be in resalable condition. Returned items must be received by Manx within 30 days. A small restocking fee may be required.

Discounts

There are special discounts available to professors, students, and consultants. A discount is also available on a "trade in" basis for users of competing systems. Call for information.

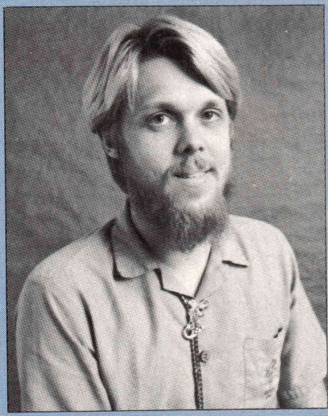
MANX

To order or for information call:

800-221-0440

Circle **no. 108** on reader service card.

RUNNING LIGHT



A new sanity seems to be infecting a growing number of computer-related companies. Could it be that our industry is becoming customer driven? Here are some examples of this new and encouraging trend:

Copy protection, long a hotly debated issue between customers and software makers, is finally being abandoned as more and more customers spurn protected software. For many years it's been difficult to find major software packages that weren't bound up in all sorts of cryptic ways. Now, with only a few exceptions, every important program is available in an unprotected form. This has been a rapid process. Why didn't it happen sooner?

The industry seems to be concerned about creating a completely machine-independent standard for the new CD-ROM technology. Everyone seems hesitant to produce any real products until there is a standard that can be applied across *all* products. We applaud this. Why didn't this kind of cooperative effort go into floppy-disk formats?

Other standards seem to be emerging as well. For example, the PostScript page-description language, developed by Adobe Systems for Apple's LaserWriter, is now supported by a new series of plug-compatible printers from Texas Instruments. This is a credit to both Apple and TI (and Adobe, of course). Other makers of page-output products have indicated their intentions to support PostScript.

Living Videotext recently made public the file-storage format for its ThinkTank series of idea-processor programs. The response from programmers was immediate and enthusiastic. We predict we'll soon see other products that are compatible with ThinkTank.

Even venerable Apple Computer is finally going to produce a Macintosh with slots. What is this world coming to?

We think the industry is finally becoming respectable. Now that there are enough customers so that a small percentage of the market share can mean a large change in dollar volume, the customer's feelings about a product are suddenly important. We encourage you to let your feelings be known. Now that companies are finally starting to respond in a big way, your voice can make a difference.

This month's hint for writers concerns communication—from us to you and back. First, we want to remind you of how important it is to include a phone number with every letter, manuscript, outline, or proposal that you send us. Often we save several weeks of correspondence by calling you directly and discussing your idea over the phone.

We have a set of form letters that we send out at various times during the article editing process. If you send us a manuscript, you will definitely get some sort of a reply within a few weeks. If your article is not rejected right away, it will be put into a file for consideration. It may spend as much as a month in there before it gets assigned or rejected. If you have sent a manuscript into *DDJ* and you haven't heard from us in a while, feel free to call me. I'll be happy to look it up and let you know what's happening.

Nick Turner

Nick Turner
editor

ARCHIVES

The Safe Bet

"Nineteen eighty-one will bring the advent of 16-bit microcomputer systems. [T]here will be a shift toward very large companies in the microcomputer market. . . . "—*Bill Gates,* InfoWorld, *January 19, 1981.*

"Microsoft played a significant role as a consultant to IBM in the development of its hardware. . . ."—*Paul Freiberger,* InfoWorld, *October 5, 1981.*

"People have been dazzled by the stuff that's gone on at Xerox PARC for years. . . . With good bit-mapped graphics and 16-bit machines on real production equipment, we can perform some of those same pieces of magic."—*Bill Gates,* InfoWorld, *January 11, 1982.*

"Insiders suggest that Apple had to go to Microsoft to get some of the [Lisa software] done. Why Microsoft? Apparently Microsoft has a couple of ex-Xerox PARC Smalltalk programmers."—*John Dvorak,* InfoWorld, *January 31, 1983.*

"There should also be some really usable portable computers using liquid-crystal displays of at least 8 lines by 40 characters."—*Bill Gates,* InfoWorld, *January 10, 1983.*

"Microsoft . . . developed all the software for the Model 100."—*Scott Mace,* InfoWorld, *April 11, 1983.*

"Most predictions [are] inside information in the guise of prognostication."—*John Dvorak,* InfoWorld, *January 10, 1983.*

Ten Years Ago in *DDJ*

"Hellman and Diffie hold that the [56-bit DES] key is not all that secure, that such a key could be broken in a day by anyone with enough money to build the trial-and-error machinery to search the 100 million billion keys . . . not too much for a government agency, say, the NSA or CIA."—DDJ, *September 1976.*

"Motorola is reliably rumored to be working on their 6809, which supposedly will give Zilog's Z80 some stiff competition. More details when we have 'em."—DDJ, *September 1976.*

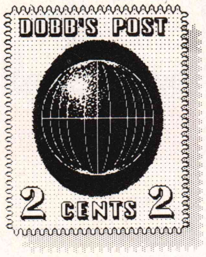
"For non-Apple systems: source and object code supplied [for a 6502 disassembler] occupies pages 8 and 9. All code is on page 8, tables on page 9. These tables may be relocated at will...code may also be relocated. Be careful if you use pages 0 or 1. Page 1 is the subroutine return stack. . . ." *Steve Wozniak,* DDJ, *September, 1976.*

"I wish to express a certain disgruntlement with some of the gargoyles that have been erected on the cathedral of LISP." —*John McCarthy,* DDJ, *September, 1976.*

"I enclose my . . . string immediate-output subroutine for 6502-based systems." —*Chris Espinosa,* DDJ, *September, 1976.*

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia
Running Light Without Overbyte

LETTERS



The Right to Optimize

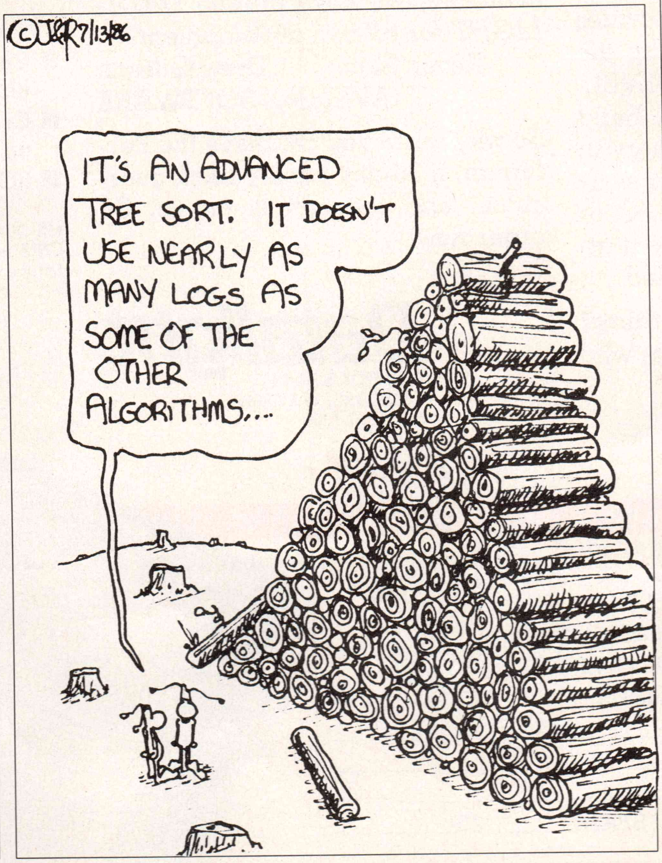
Dear *DDJ*,

Richard Campbell's March column on NS32016 square root calculations prompted me to implement his algorithm on my 68000 system and do some comparative measurements. My system is a 68000 board hooked to an Apple II computer running at an effective clock rate of approximately 8.1 MHz. (The nominal clock rate of 12.5 MHz is slowed down by one additional wait cycle on every bus transfer and a software D-RAM refresh scheme.)

Unfortunately, I do not have a C compiler for my board, so I had to resort to rewriting Mr. Campbell's program for what I have available, which is a UCSD p-code System adapted from Apple Pascal to the 68000. One of the limitations of this system is that it handles only 16-bit signed integers, so I had to use some routines of my own to support 32-bit integer arithmetic in order to stay compatible with Mr. Campbell's benchmarks, which used a loop from 0 to 60,000 to time the square root routines.

As the Pascal compiler produces p-code object programs, I wrote the square root routine in 68000 assembly language and linked it as an external procedure to the surrounding Pascal program [Listing One, page 56]. For performance measurements, I used a system *time* procedure with a resolution of 1/60th of a second and, like Mr. Campbell in his article, a loop performing 60,001 calls to the square root routine with arguments ranging from 0 through 60,000. To derive the time it takes for the square root routine to execute, I measured the execution time of the Pascal program and then subtracted from it the execution time of the same program linked to a routine that skips the calculation of the square root; the latter time therefore represents the overhead for the loop and the procedure call mechanism, which has nothing to do with the square root algorithm itself.

Using the routine that mimics Mr. Campbell's compiler-generated code [Listing Two, page 56], the program executes in 16.73 (+/− 0.02) seconds on my system (without printing or counting shifts and divisions) and the empty shell program needs 12.08 seconds. Calculating 60,001 square roots, therefore, takes 4.65 seconds, for an average of 77.5 microseconds per square root. To answer Mr. Campbell's question about whether hand optimizing the assembly code is worth it, I did just that. Using the same measuring procedure as above, the optimized code [Listing Three, page 56] needed 3.90 seconds to perform the 60,001 square root calculations (without overhead), for an average of 65 microseconds per square root—a speedup of about 16 percent. In my book, that is well worth the little trouble it takes to optimize the routine, especially if it is used often.



To find out by how much Mr. Campbell's algorithm is faster than the algorithm mentioned by Jim Cathey, I also performed the measurements for a program using that algorithm to calculate the square root [Listing Four, page 58]. Without overhead, Mr. Cathey's algorithm took 8.66 seconds to perform the 60,001 square roots, or an average of 144 microseconds per square root; it is therefore almost twice as slow as the unoptimized code for Mr. Campbell's algorithm.

To compare my results for the MC68000 with the published ones for a 6-MHz NS32016, they should be multiplied by a factor of 1.60 (9.6 MHz/6 MHz) to get timings equivalent to a 6-MHz 68000. If this is done, the time to calculate a square root using the unoptimized code becomes 124 microseconds, the optimized version needs 104 microseconds, and the square root calculation using bit shifting needs 231 microseconds. This clearly disproves Mr. Campbell's statements about the superiority of the NS32016.

Regarding Mr. Campbell's comments about the bit-shifting algorithm "hitting the NS32016 below the belt," I can only say that Mr. Cathey's algorithm was not specifically designed to do that but is rather a general-purpose algorithm that merely points out certain weaknesses in the NS32016's design. In my opinion, the MC680XX family is clearly superior to the microprocessors of the NS320XX family; everything you can do with a NS32016, you can also do on a 68000—often more easily,

(continued from page 10)

usually faster, and then some, as illustrated above.

Other than that, I agree with Mr. Campbell's conclusion that many bit-level tricks of the 8-bit era are outdated and should be replaced by newer and faster algorithms more appropriate for 16/32-bit processors. I hope to find exactly such articles in future issues of *DDJ*.

Thomas Wieland
8676 Anthony, #5
Pierrefonds, PQ H84 2B6
Canada

8080 Simulator

Dear *DDJ*,

Hats off to Jim Cathey for such a nice and useful piece of MC68000 code. [See "COM: An 8080 Simulator for the MC68000," January 1986.] After I had typed in the code (only one typo and an easy to find one at that), I couldn't resist adding Z80 relative jumps and *djnz* [Listing Five, page 58].

In an attempt to improve on the simulator's speed, and remembering an old trick from FORTH compiler writing days, I expanded *mloop*, which I duly dubbed *NEXT*, at the end of every opcode simulation, thereby gaining eight t-cycles per simulated 8080 instruction as well as freeing *a6* for other purposes (I promptly used it to point to *pseudo-HL*).

If your system is configured with CP/M-68K above the TPA in such a way that the simulator runs below $8000, you can even go a step further by having a jump table (*optabl*) in the shape of *dc.ws* and a *NEXT* that looks like that shown in Table 1, left.

```
moveq    #0,d0                 * clr hi word
move.b   (pseudopc)+,d0        * fetch next opcode
add.w    d0,d0                 * d0 <-- offset into word table
move.w   0(opptr,d0.w),a0      * d0 max is $ff << 1 ($1fe)
jmp      (a0)                  * exec opcode
```

Table 1: *Improved NEXT routine for 8080 simulator*

Together with various minor alterations too voluminous to present here, this last modification sped up the simulator to such an extent that it now runs slightly faster than SoftDesign's Z80 emulator, alas without providing full Z80 support. As many CP/M-80 programs don't use the extra Z80 instructions anyway, this restriction is seldom felt.

In order to spare other CP/M-68K users a set of blistered fingers, I will gladly copy the source for anyone who provides a formatted 8-inch or 5¼-inch floppy disk (please give fcb and skew details if not 8-inch SSSD).

Edmund Ramm
Postfach 38
D-2358 Kaltenkirchen
West Germany

Corrections

Dear *DDJ*,

The C implementation of Bresenham's line-drawing algorithm that appeared in the May 1986 issue contains several errors. [See "Simple Plots with the Enhanced Graphics Adapter" by Nabajyoti Barkakati.] These errors cause the routine *v_draw* to fail in the first and third quadrants. I have enclosed a corrected version, *c_draw* [Listing Six, page 62].

In *v_draw*, a statement on line 75 is missing altogether. This causes the routine to break in a not so subtle manner; I assume the omission is a misprint. However, initialization of the decision variable *d* (lines 34 and 80) is incomplete, resulting in somewhat more obscure problems. The value to which *d* must be initialized depends upon whether the line slope is positive or negative. If the initialization is handled incorrectly, symmetry is destroyed and endpoint overrun errors are introduced. These problems may not show up unless the routine is tested at all the boundary conditions.

The line-drawing routine runs rather slowly when the BIOS point-plotting routine is used, even on an IBM PC/AT. Possibly the use of Bruce Smith's assembly-language point-plotting algorithm, published in the January 1985 16-Bit Software Toolbox, would speed things up.

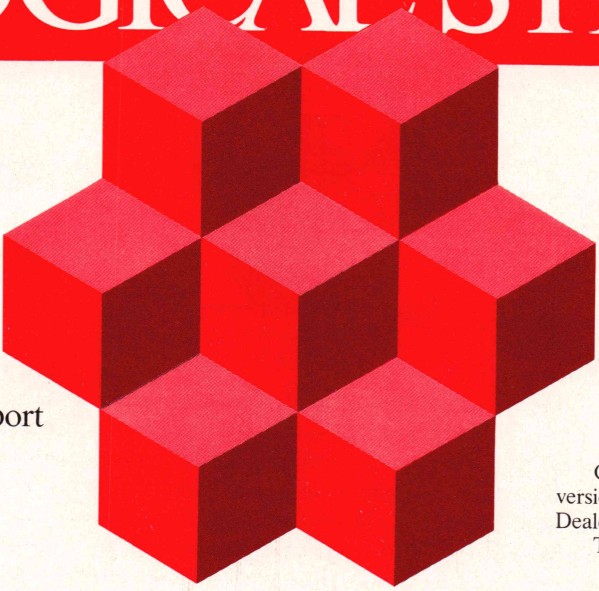
Joseph N. Mente
Titanic CodeWorks
916 Olive Rd.
Homewood, IL 60430

Listing One of Everett Carter's article, "Forth Goes to Sea," (July 1986) was truncated. The rest of the listing begins on page 50.

DDJ

(Listings begin on page 56.)

TAKE THE NEXT LOGICAL STEP



$89 Price

- Separate Compilation
- Native Code Generation
- Large Memory Model Support
- Multitasking
- Powerful Debugging Tools
- Comprehensive Module Library
- Available for the PC and the VAX

MODULA-2/86

NEW, improved Turbo Pascal to Modula-2 Translator!

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.
To place an order call our special toll free number:
800-231-7717
In California:
800-552-8885

Move up to LOGITECH MODULA-2/86. Whether you're a single programmer or part of a team, with LOGITECH MODULA-2/86 you'll decrease your overall development cycle and produce more reliable, more maintainable code. Build your program using our extensive library modules, your own modules or those from a growing list of available third-party software vendors. If you're a Turbo Pascal user you can even take your existing code along with you with the help of our new Translator!

NEW & IMPROVED!

Turbo Pascal to Modula-2 Translator $49

Now it's even easier for Turbo users to step up to Modula-2/86. Our improved Translator changes your Turbo source code into Modula-2/86 source, solving all the incompatibilities, and translating the function calls of Turbo into Modula-2/86 procedures. Implements the complete Turbo library!

LOGITECH MODULA-2/86 $89

Complete with Editor, Run Time System, Linker, 8087 Software Emulation, Binary Coded Decimal (BCD) Module, Logitech's comprehensive library, Utility to generate standard .EXE files. AND more!

- LOGITECH MODULA-2/86 with 8087 Support $129
- LOGITECH MODULA 2/86 PLUS $189

For machines with 512K of RAM. Takes advantage of larger memory to increase compilation speed by 50%.

Turbo Pascal is a registered trademark of Borland International.

RUN TIME DEBUGGER $69

(Source level!)
The ultimate professional's tool! Display source code, data, procedure call chain and raw memory. Set break points, assign values to variables, pinpoint bugs in your source.

UTILITIES PACKAGE $49

Features a post-mortem debugger (PMD). If your program crashes at run time the PMD freezes the situation so you can pinpoint, in the source, the cause of the error and the status of the data. Also includes a disassembler, cross reference utility and version that allows conditional compilation.

LIBRARY SOURCES $99

Source code for our major library modules is now available for customization or exemplification.

WINDOW PACKAGE $49

Now you can build true windowing into your Modula-2 code. Powerful, though only 15K in size. Features virtual screens, color support, overlapping windows and a variety of borders.

MAKE UTILITY $29

Automatically selects modules affected by code changes to minimize recompilation and relinking. Even figures out dependencies for you!

CROSS RUN TIME DEBUGGER AND ROM PACKAGE $199

Now available at an introductory price!

YES I want to move up to LOGITECH MODULA-2/86!

Here's the configuration I'd like:

☐ Logitech Modula-2/86	$89
☐ with 8087 support	$129
☐ Plus Package	$189
☐ Turbo to Modula Translator	$49
☐ Run Time Debugger	$69
☐ Utilities Package	$49
☐ Library Sources	$99
☐ Window Package	$49
☐ Make Utility	$29
☐ ROM Package	$199

Total Enclosed $______

☐ Visa ☐ Mastercard ☐ Check Enclosed

Card Number ______ Expiration Date ______

Signature ______

Name ______

Address ______

City, State ______

Zip ______ Phone ______

LOGITECH

Logitech, Inc.
805 Veterans Blvd.
Redwood City, CA 94063
Telephone 415-365-9852
For European pricing please contact:
Logitech SA
Box 32, CH-1143
Apples, Switzerland
Telephone 41-21-774545

Circle **no. 257** on reader service card.

C CHEST

Directory Traversal, Trailing ^Zs, and Horrifying Experiences

Directory Traversal

This month I'm going to look at another sort of tree—a directory tree. The program presented here does several things, depending on how it's invoked. If the program is named whereis.exe, then the command *whereis <fname>* will search the entire directory tree for a file called fname, printing the full path name of the file when it's found. Wildcard characters are recognized in the name (as in *whereis *.c*). Be careful if you're running this program under the shell. You'll have to escape the wildcards to prevent the shell from expanding them (*whereis "*.c"* or *whereis *.c*).

By renaming the same program to dtree.exe, several functions are added. The basic command syntax is:

```
dtree [root] [-f<name>] [-s] [-d]
                         [-e cmd arg . . .]
```

The root indicates where to begin the directory traversal. If it's not specified, then / is used. Thus, *dtree* (with no arguments) descends recursively through the entire directory tree, starting from the root directory, printing a list of all the directories on your disk. *Dtree <dname>* prints a list of all directories at or below *<dname>* in the directory tree. So *dtree /* works in the same way as

by Allen Holub

dtree without arguments does. *Dtree /src* prints a list of all subdirectories below */src* (including the subdirectories of the subdirectories). Table 1 , page 16, shows a sample output from my own disk.

The *−d* command-line switch causes a graphic representation of the directory tree to be printed rather than a simple list of the names. If *−s* is specified too, short names are used instead of the full path name. An example is given in Table 2, page 16. When the program's output is redirected to a file, it's printed as shown (with plus signs and dashes). If output goes to *stdout*, the IBM box-drawing characters are used so that you have a prettier picture on the screen.

The *−e* switch is used to execute commands. All command-line arguments that follow *−e* are taken as a command that will be executed from each directory as it's visited. A space must follow the *e*. For example:

```
dtree / -e ls -l
```

prints a list of every file in every directory on the disk using the *ls* long format (that is, because the *−l* follows the *−e* on the command line, it's passed to *ls* rather than being interpreted by *dtree*). The directory name is printed too (just before the command is executed). If you need to execute either a batch file or a command that's part of the shell, you have to invoke the shell explicitly. Examples are:

```
dtree / -e command /c dir
dtree / -e command /c batchfile
                        arg arg arg
dtree / -e sh batchfile arg arg arg
dtree . -e sh cp "*.c" a:
```

The first example does the same as *dtree −e ls* does except that the *dir* command (that's internal to COMMAND.COM) is used. The second example executes an MS-DOS batch file; the arguments following the file name are passed to the batch file. The third example does the same, but my own shell (as distributed by *DDJ*) is used rather than COMMAND.COM. The last example backs up all .c files in an entire system—that is, it copies all .c files in the current directory and any subdirectories to the a: drive. The explicit invocation of the shell is necessary because wildcard expansion must be done in each directory as it's visited, and this expansion is done by the shell, not by *cp*. Had you said:

```
dtree . -e cp *.c a:
```

the **.c* would have been expanded by the shell to all files in the current directory that end in *.c*. *Dtree* would copy these and then descend to a subdirectory and try to find files having the same names as the ones in the parent (because the **.c* is expanded by the shell before *dtree* ever sees the command line).

The switch *−f<name>* (no space between *f* and *<name>*) causes the program to search for the file called <name>. So *dtree / −f<name>* does the same thing as *whereis <name>* does. Using *dtree* rather than *whereis* has two advantages. First, *dtree* can start the search at any directory, but *whereis* always starts at the root. So:

```
dtree /src -ffoo.c
```

finds foo.c only if it exists at or below */src* in the directory tree. The second advantage comes from using *−f* and *−e* together on a single command line. Here, the command is executed only if the file is found. For example, if you're running under sh (rather than COMMAND.COM) using the rm program provided with the /util package to remove files:

```
dtree / -ffoo -e rm foo
```

Finds all occurrences of the file foo

C CHEST
(continued from page 14)

on the disk and deletes them, and:

```
dtree /src/trees "-f *.bak" -e
                      sh -c rm "*.bak"
```

deletes all .bak files in the */src/trees* directory (and any subdirectories of */src/trees*). The *−f* argument has to be quoted to prevent the shell from expanding it (*dtree* expands the wildcards itself and *−f* can take only one argument). Similarly, rm doesn't expand wildcards so you must invoke a shell to do it. The **.bak* has to be quoted so that the subshell will do the expansion instead of the current shell.

Because the *delete* and *copy* functions are built into COMMAND.COM, you can say:

```
dtree /src -f *.bak -e command /c
                        copy *.c a:
```

to do the equivalent operation from outside the shell. You don't have to quote any arguments because COMMAND.COM won't expand them; *copy*, on the other hand, will.

```
/src
/src/getargs
/src/grep
/src/nr
/src/nr/hyphen
/src/red
/src/shell
/src/shell/util
/src/shell/util2
/src/sm
/src/small-c
/src/sml-tool
/src/tools
/src/tools/doc
/src/tools/lattice
/src/tools/old
/src/tools/trees
/src/tools/trees/old
/src/util
```

Table 1: *Output from* dtree /src

```
src
+-----getargs
+-----grep
+-----nr
|          +-----hyphen
|
+-----red
+-----shell
|          +-----util
|          +-----util2
|
+-----sm
+-----small-c
+-----sml-tool
+-----tools
|          +-----doc
|          +-----lattice
|          +-----old
|          +-----trees
|                     +-----old
|
|
+-----util
```

Table 2: *Output from* dtree / src −d−s>file

Dtree.c

The source for *dtree( )* is in Listing One (page 62). The modus operandi of the program is determined in *main( )* on lines 380−394. *Main( )* examines *argv[0]* to see by what name it was invoked. If *whereis* is used, the *if* clause is executed; if anything other than *whereis* is used, then the *else* clause is executed. Note that *argv[0]* isn't supported by DOS, Version 2.x, so this automatic configuration will work only if you're using DOS 3.x or if you're running the shell (the *reargv( )* function will give you access to *argv[0]*).

The *main( )* routine also determines which character set to use for tree printing (on line 397). It does this in the same way as does the *print* routine I discussed last month. The same limitations apply, too (that is, graphics are used unless standard output is redirected to a file). The *signal( )* call on line 405 is necessary because the traversal algorithm actually changes the current directory as it traverses. The *signal( )* call sets up the interrupt system so that when a ^C is encountered, the current directory will be set back to whatever directory you started from when the program booted. The actual resetting is done in *onintr( )* on lines 361−366.

Dtree( ) can't use *getargs* because it uses a position-dependent command-line switch (*−e*). So the program's arguments are processed by hand in *doargs( )* on lines 270−332. It works pretty much the same as *getargs( )* does, setting global variables to correspond to encountered command-line switches and compressing *argv* to remove all command-line switches. Note that all *argv* entries following *−e* on the command line aren't analyzed (that's the test for *Args* on line 295; *Args* is NULL if a *−e* hasn't been found).

The basic traversal algorithm is done by *prnt( )* (on lines 174−232). It uses a modified version of the *preorder* tree-printing algorithm that I discussed two months ago, so I won't cover that material again. However, instead of using a basic tree-traversal algorithm:

```
trav( root )
{
    print( root );
    trav ( left );
    trav ( right );
}
```

which can be restated for multiway tree traversal as:

```
trav( root )
{
    print( root );
    for( each child )
        trav( child );
}
```

dtree uses:

```
trav( current directory )
{
    print( current directory's name );
    get list of all subdirectories;
    for( each subdirectory )
        trav( subdirectory );
}
```

The code on lines 188−192 just prints vertical bars in the appropriate places, provided you're printing a picture of the tree (that is, *−d* is specified on the command line, causing *Draw* to be true). If you're searching for a file, the *if* statement on lines 194−198 is executed. The directory name is not printed. Rather, if the file exists, you execute the command tail. *Execute( )* won't do anything if *−e* isn't found on the command line. If you're not looking for a specific file, the *else* clause on lines 199−203 prints the current directory name and then executes the command tail.

Prnt( ) gets the list of subdirectories using the same *dir( )* subroutine that's used by the shell (the code for all these routines and for the mydir.h file that's *#include*d on line 4 are in

C CHEST
(continued from page 16)

the March 1986 C Chest. *Mk_dir( )* creates a *DIRECTORY* structure on line 205. This structure is initialized on lines 211–215 so that *dir( )* will get a sorted list of subdirectories, using the full path name of each directory in the list. *Dir( )* itself is called on line 219; the recursive traversal of each subdirectory is done by the *while* loop on lines 221–225; and memory used by the the *DIRECTORY* structure is freed on line 230 [with the *del _dir( )* call].

Find( ) (on lines 137–170) uses the same directory routines to look for a file. It will expand any wildcard characters in the file spec and print the whole list of matching files. The remainder of the code you've already seen, either last month or the month before, so I won't cover it again here.

Fixing ^Z-Terminated Files

Back in May, Ray Duncan mentioned a problem with ^Z-terminated files being read by the Microsoft Macro Assembler, Version 4.0 (16-Bit Software Toolbox, p.109). The *include* function ignores the end-of-file mark (^Z) and uses the file size, so it will kick out error messages if you create a file with any editor that terminates its files with ^Zs rather than just making the files the correct length. Mince, Vedit, and WordStar (in document mode) are three of these editors. You often see this sort of padding in programs that were originally written for CP/M or MS-DOS, Version 1 (where ^Z is the only way to end a file). Microsoft has been trying to get rid of the ^Z end-of-file marker since it introduced Version 2 of MS-DOS, and I guess it's finally playing hardball.

This ^Z problem is responsible for a lot of weird behavior on the part of the shell, too. The formatted I/O routines in Version 3.0 of the Microsoft C compiler get hopelessly confused when they try to read a ^Z-terminated file. So if you try to execute a Vedit-created batch file from the shell, strange things start happening (characters mysteriously appear and disappear, error messages such as "stdout: no room left on device" are printed, and so on).

Unfortunately, the fix that Ray gave in his column (enter, write, and then exit from EDLIN) doesn't seem to work all the time (or so say friends who've tried it), and it turns out to be almost impossible to fix this problem in Microsoft-compiled programs without going to unformatted binary input (that is, *open( )*, *read( )*, and so on). So I wrote a utility called fix to deal with it (see Listing Two, page 68).

Fix removes all trailing ^Zs from a text file (actually it truncates the file at the first ^Z it finds). Usage is:

```
fix file [file . . . ]
```

where the command line holds a list of files to fix. The program renames the files specified on the input line to xxx.bak, where xxx is the original root part of the file name. It then overwrites the original file, removing the trailing ^Zs.

The program is very short and pretty much self-explanatory. Note that the *ctlc( )* and *reargv( )* calls on lines 54 and 55 are useful only if you're running under the shell and they're supplied along with the shell.

First "language specific" DBMS written exclusively for C applications is also royalty free

"db_VISTA™ lets you easily build complex databases with many interconnected record types. . ."
Dave Schmitt, President, Lattice, Inc.

Designed exclusively for C, db_VISTA™ is a language specific database management system. Both single and multi-user versions let you take full advantage of C, through ease of use, transportability and efficiency.

Every Line of Code Written in C for C Programmers

All functions use C conventions so you will find db_VISTA easy to learn. db_VISTA operates on most popular computers, and because it is written in C it can easily be ported to most computers.

Royalty-Free, You Only Pay Once

Whether you're developing applications for a few customers, or for thousands, the price of db_VISTA is the same. If you are currently paying royalties for a competitor's database, consider switching to db_VISTA and say goodbye to royalties. To help you make the change over to db_VISTA, ASCII file transfer utilities are included. dBASE file transfer utilities are available as an option.

More from your database applications with source code

Source code includes all db_VISTA libraries and utilities.

1. Recompile our run-time libraries utilizing non-standard compiler options.
2. Create a debugging library including a function traceback by activating preprocessor commands embedded in the source code.

Multi-user and LAN capability

Information often needs to be shared. db_VISTA has multi-user capability and supports simultaneous users in either multi-tasking or local area networking environments, allowing the same C applications to run under UNIX and MS-DOS.

Faster execution without data redundancy

Less data redundancy means reducing disk storage requirements and maximizing data access performance. A customer evaluating a leading competitor's product prior to purchasing db_VISTA benchmarked db_VISTA's retrieval time to be 276% faster than a leading competitor.

Complete documentation included

User manual contains 193 pages, 8 diagrams, 10 tables, appendices, an extensive index, plus a database application example. 9 chapters with complete instructions.

db_QUERY™ lets you ask more of your database

db_QUERY is a linkable, SQL-based ad hoc query and report writing facility. It's also royalty-free.

Circle **no. 206** on reader service card.

30-Day Money-Back Guarantee

We wish to give you the opportunity to try db_VISTA for 30 days in your development environment and if not satisfied return it for a full refund.

Price Schedule

	db_VISTA	db_QUERY
Single-user	$195	$195
Single-user with Source	$495	$495
Multi-user	$495	$495
Multi-user with Source	$990	$990

FREE 60 Days Application Development Support
All software Not Copy Protected

Call Toll-Free Today!

To order or for information, call TOLL-FREE at 1-800-843-3313, then at the tone touch 700-992 or call 206-747-5570.

VISA and MASTERCARD Accepted

Read what others say about db_VISTA

"If you are looking for a sophisticated C programmers database, db_VISTA is it. In either a single or multi-user environment, db_VISTA lets you easily build complex databases with many interconnected record types. The multi-user implementation handles data efficiently with a LAN and Raima's customer support and documentation is excellent. Source code availability and a royalty-free run-time is a big plus."
Dave Schmitt, President Lattice, Inc.

"Not 'yet another user-friendly database', it is a DBMS aimed at the technical C programmer instead of the non-technical end-user."
Hal Schoolcraft, Data Based Advisor March, 1985

"On the whole, I have found db_VISTA easy to use, very fast with a key find, and powerful enough for any DBMS use I can imagine on a microcomputer."
Michael Wilson, Computer Language September, 1985

db_VISTA Version 2.11 Database Management System for C

Database Record and File Sizes

- Maximum record length limited only by accessible RAM
- Maximum records per file is 16,777,215
- No limit on number of records or set types
- Maximum file size limited only by available disk storage
- Maximum of 255 index and data files

Keys and Sets

- Key length maximum 246 bytes
- No limit on maximum number of key fields per record—any or all fields may be keys with the option of making each key unique or duplicate
- No limit on maximum number of fields per record, sets per database, or sort fields per set
- No limit on maximum number of member record types per set

Utilities

- Database definition language processor
- Interactive database access utility
- Database consistency check utility
- Database initialization utility
- Multi-user file locks clear utility
- Key file build utility
- Data field alignment check utility
- Database dictionary print utility
- Key file dump utility
- ASCII file import and export utility

Features

- Multi-user support allows flexibility to run on a local area network
- File structure is based on the B-tree indexing method and the network database model
- Run-time size, variable—will run in as little as 64K, recommended RAM size is 256K
- Transaction processing assures multi-user database consistency
- File locking support provides read and write locks on shared databases
- SQL-based db_QUERY is linkable
- File transfer utilities included for ASCII, dBASE optional

Operating System & Compiler Support

- Operating system's MS-DOS, PC-DOS, Unix, Xenix, Macintosh and Amiga
- C compiler's Lattice, Microsoft, DeSmet, Aztec, Computer Innovations, Xenix and Unix

Independent Benchmark Results

Eleven key retrieval tests on sequentially and randomly created key files. Benchmark procedure adapted from "Benchmarking Database Systems: A Systematic Approach" by Bitton, DeWitt, and Turbyfill, December, 1983.

Total Retrieval Time of 11 Tests	
db_VISTA	:671.24
Leading Competitor	:1,856.43

RAIMA™ CORPORATION
12201 S.E. Tenth Street
Bellevue, WA 98005 USA
(206) 747-5570
Telex: 9103330300 BCN RIVERTON

1 (800) 843-3313
at the tone touch 700-992

MasterCard VISA

(continued from page 18)

Don't include these calls if you're using COMMAND.COM.

A Tale of Woe

My friend Bill Wong, who lives across the street, has gallantly volunteered to beta test the shell as I bring up the new version. He was having problems with it the other day, and we thought these problems might be DOS-related (he's running Version 3.0 rather than 3.1), so we decided to upgrade the DOS version on his hard disk. Earlier versions of DOS (pre-3.1) made it impossible to create a system disk except by formatting the disk with the /s flag set (*copy* won't copy system files). This, of course, forced you to back up, format, and then restore your entire hard disk to install a new version of DOS. Version 3.1 comes with a utility called sys that can transfer a system to a hard disk, but I had forgotten that this utility existed.

So, I tried to install the new DOS version on Bill's disk by removing the hidden and system attributes from IBMBIO.COM and IBMDOS.COM and then copying them over to the hard disk in the normal way. I've a program (called chmod) that can change these attributes (*attrib* can change only the read-only attribute). Files successfully transferred, I pressed Ctl-Alt-Del, and to my horror, the system wouldn't boot. It didn't even print an error message—it just hung. "Oh God," I said to myself, "I've destroyed Bill's hard disk." Fortunately, things weren't quite that bad. The system booted from the floppy with no trouble. We could at least get the thing running again to try to find out what the problem was.

To make a long story short, we fooled around for more than an hour, trying everything we could think of (including running the sys program, now rediscovered by us). Nothing worked. No amount of copying fixed anything. Changing attributes didn't help. Sys said that it had transferred the system, but it still wouldn't boot from the hard disk. So, in desperation we deleted the entire root directory (including the system files) and tried to run sys again. Now it started giving us "No room for system on destination disk" error messages. The disk was only two-thirds full, so there was obviously some other problem.

Finally I threw in the towel and decided to call Microsoft. It told me that it does not support end-users of DOS and that I'd have to call IBM. IBM told me that it does not support end-users of DOS and that I'd have to call the store that sold me the computer. ComputerLand told me that it had no idea what the problem was, though it could guess. The person I talked to suggested reading the *DOS Technical Reference*. So, it seems there is no technical support available for MS-DOS—from anyone.

I dug out the *Technical Reference* and actually did find something there. It was even in the index (under *system reset*). When MS-DOS boots: "The boot record then checks the root directory to assure that the first two files are IBMBIO.COM and IBMDOS .COM. These two files must be the first two files, and they must be in that order (IBMBIO.COM first, with its sectors in continuous order)." So, my theory is that when you open a file for overwrite, DOS frees up the associated *FAT* entries and directory space. That is, it doesn't just overwrite an existing file with new data; it throws away the existing file and then starts from scratch, putting the copy wherever it feels like. So the two boot files ended up in the wrong place. Meanwhile all our thrashing around on the disk ended up putting something into the sectors that IBMBIO.COM wanted to occupy. This explains sys' inappropriate error message. There's plenty of room on the disk, but there's probably no room on track 0. I still don't understand why sys told us that the system had been transferred successfully the first few times we tried to use it. I suspect this has something to do with directory entries for IBMBIO.COM and IBMDOS.COM already existing but pointing at the wrong place on the disk. It seems to me that sys should check for things such as that. It obviously doesn't. I also don't understand, if the positions of IBMBIO.COM and IBMDOS.COM on the disk are so important, why you can delete them at all. To my mind they shouldn't even be in the directory system because you can't treat them like you can any other file.

I'm sure there's a lesson to be learned from all this, but I'm not sure what it is. Your only recourse is to back up, reformat the hard disk, and restore. Looked at in a truly Polyanna fashion, at least disk accesses will speed up because all the file fragmentation will be eliminated.

To add injury to insult, bringing up Version 3.1 didn't fix the problem with the shell (which turned out to be the ^Z problem that I described earlier). Aren't computers wonderful?

Availability

The listings from this month are all available on CompuServe (type go ddj). The chmod program, used to change file attributes, is now distributed by *DDJ* as part of Version 1.1 of the /util package. This package also includes the rm and cp utilities used in the examples. If you have an earlier version, you can get an upgrade from *DDJ* for $6. The fix program is distributed with Version 2.0 of the shell (available from *DDJ*, upgrades are $6, too) and is also available on CompuServe. The directory-related routines used by dtree.c are also part of the shell. They were originally published in this column in March 1986 (the listing begins on page 56 of that issue). Note that the mydir.h file on the shell distribution disk is called dir.h in that listing.

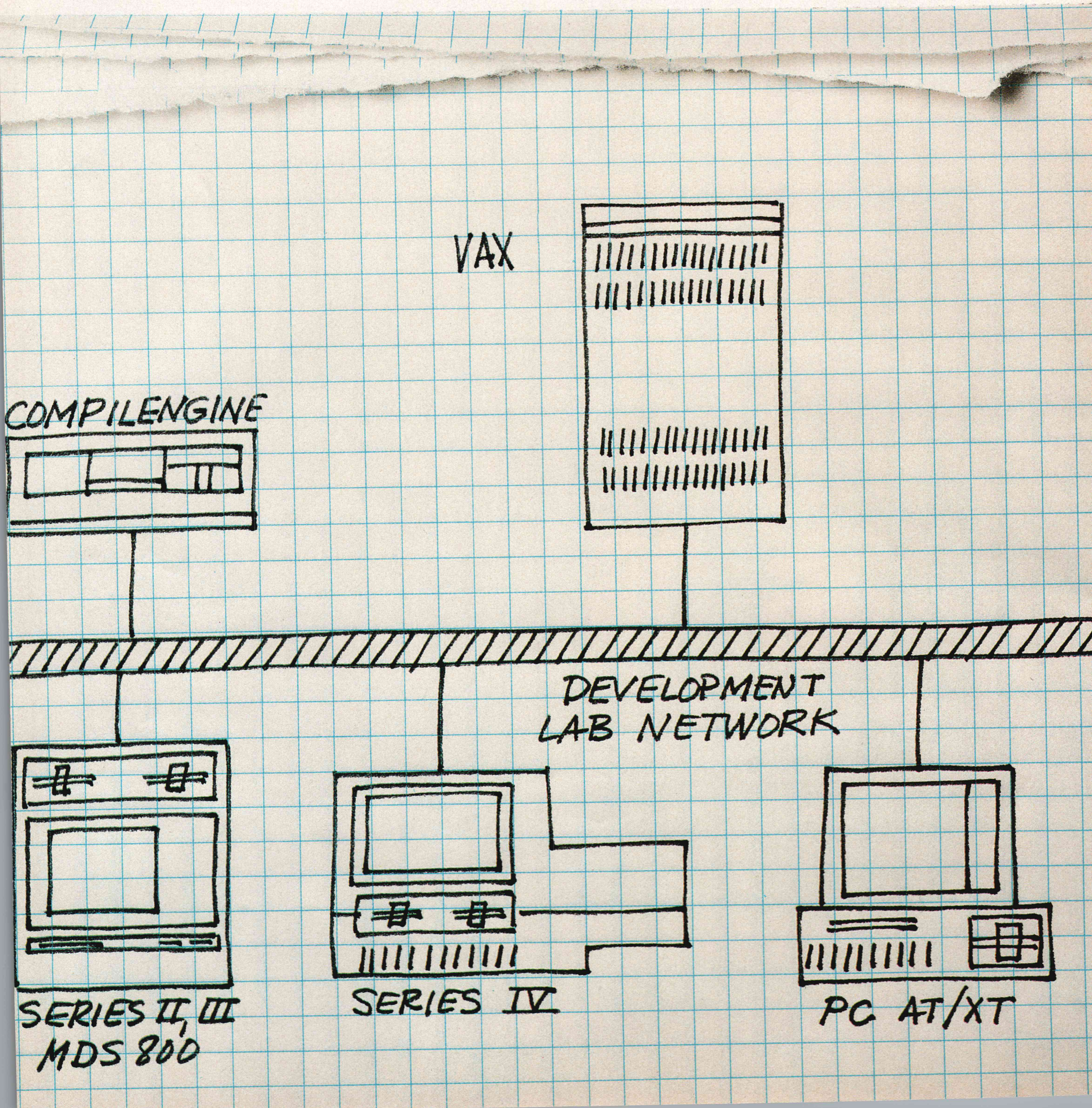
An IBM PC-compatible disk containing the complete sources for dtree.c and fix.c (including the directory routines) is available for $25 from Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705.

DDJ

(Listings begin on page 62.)

Vote for your favorite feature/article.
Circle Reader Service **No. 1**.

NOBODY ELSE HAS MADE THE CONNECTION.



There's only one way to link the extensive resources of Intel's microprocessor development system with the power of a DEC VAX® and the affordability of the IBM® PC.

And that's via our enhanced version of OpenNET™ for the development lab.

Thanks to this open architecture network, engineers can have immediate, and transparent, access to other team members' work.

Plus you have the ability to add special-purpose hardware to make those teams more productive.

For example, by connecting our high-performance file server, the Network Resource Manager, you can off-load file management and job distribution from the shoulders of your design team. On the off chance nobody wants to spend his time chasing down floppy disks.

And then you can hook up an 80286-based Compilengine to off-load compute-bound compilations from VAXs and workstations. Leaving them, and their human partners, more time for interactive tasks.

Equally important, the network maximizes the value of your existing development hardware while minimizing your outlay for new equipment.

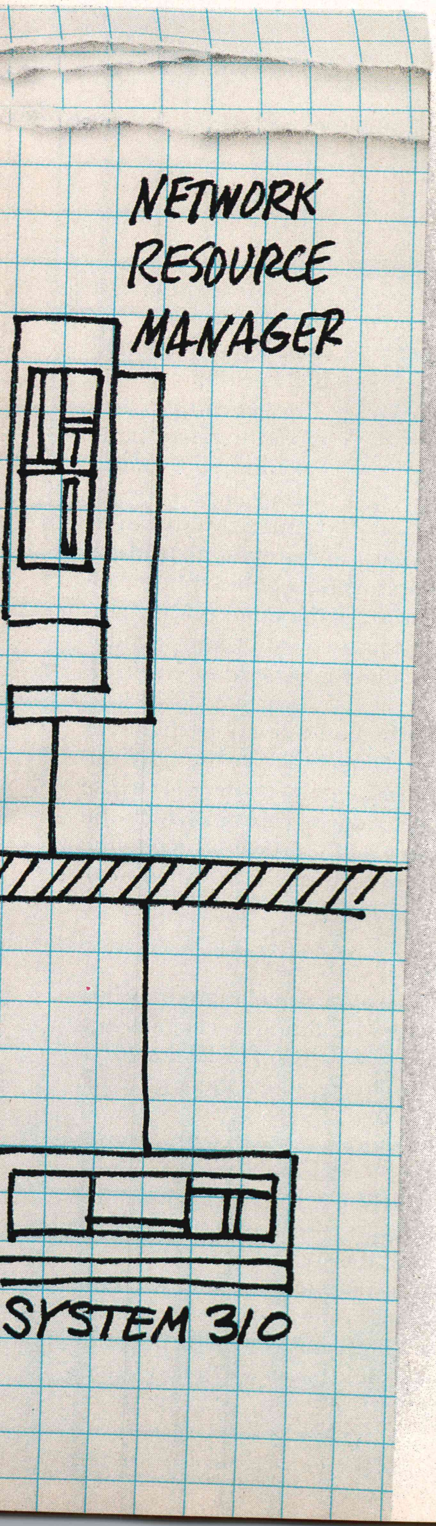
That's because OpenNET adheres very rigidly to some very flexible standards. Standards like Ethernet/IEEE 802.3. And the ISO message delivery and Intel/IBM/Microsoft® Network File Access protocols.

All of which means existing development hosts, languages and tools, including ICE, are instantly compatible with the latest ones. So you can avoid obsoleting one set of tools just to use another.

We'll even take full responsibility for servicing and supporting your network. Anywhere in the world.

Sound like we've got things together? Then call us at (800) 548-4725.

Ask to meet with one of our experienced network engineers. We'll see you make all the right connections.



intel®

Circle **no. 179** on reader service card.

Come see us at DEXPO-West; Booth #1132

Curve Fitting with Cubic Splines

by Ian E. Ashdown

The program can generate a curve for a thousand or more data points.

Before computer-aided drafting workstations completely replace the draftsman's pencil and paper, let's examine one of the draftsman's tools: the spline. Presented with data in the form of points on an x−y plane, the draftsman uses a spline—a flexible strip of metal or plastic—to draw a smooth curve between them.

The technique is very simple. After plotting the data on a sheet of paper, an appropriately sized spline is held in place at these points (referred to as "knots") with weights or pins. The draftsman then traces the curve formed by the spline. For any given set of knots, the curve generated is independent of the spline chosen and is thus exactly reproducible.

From mechanical engineering, elementary beam theory shows that if the spline is not too severely stressed, it will conform to a curve described by a set of cubic polynomial equations, one between each pair of adjacent knots. Adjacent polynomials meet at their common endpoints (the knots), and their slopes and rates of curvature at these points are equal. Stated in mathematical terms, these polynomials join continuously at the knots with continuous first and second derivatives.

Knowing this, you can develop a mathematical model of the draftsman's splines and from this model construct a computer program for interpolating a smooth curve between a set of knots. With a bit of care in choosing algorithms, such a program can quickly and accurately generate a curve for a thousand or more data points on the smallest of personal computers. It can even be adapted to interpolate a smooth surface between points plotted in three dimensions.

Developing the model involves basic calculus and matrix theory. If you are unfamiliar with such mathematics, rest assured that the resultant algorithms are very easy to program and using a cubic spline program requires no understanding of the underlying mathematical theory. Give the program a set of knots, and it will dutifully interpolate a smooth curve in all (well, almost) cases.

Why then discuss the mathematics of cubic splines at all? There are two answers. One is that seeing how the algorithms are developed gives you the confidence to use them. The other is that there may be cases in which the algorithms will not perform exactly as desired. Knowing their theory may enable you to create a modified algorithm to fit the problem at hand.

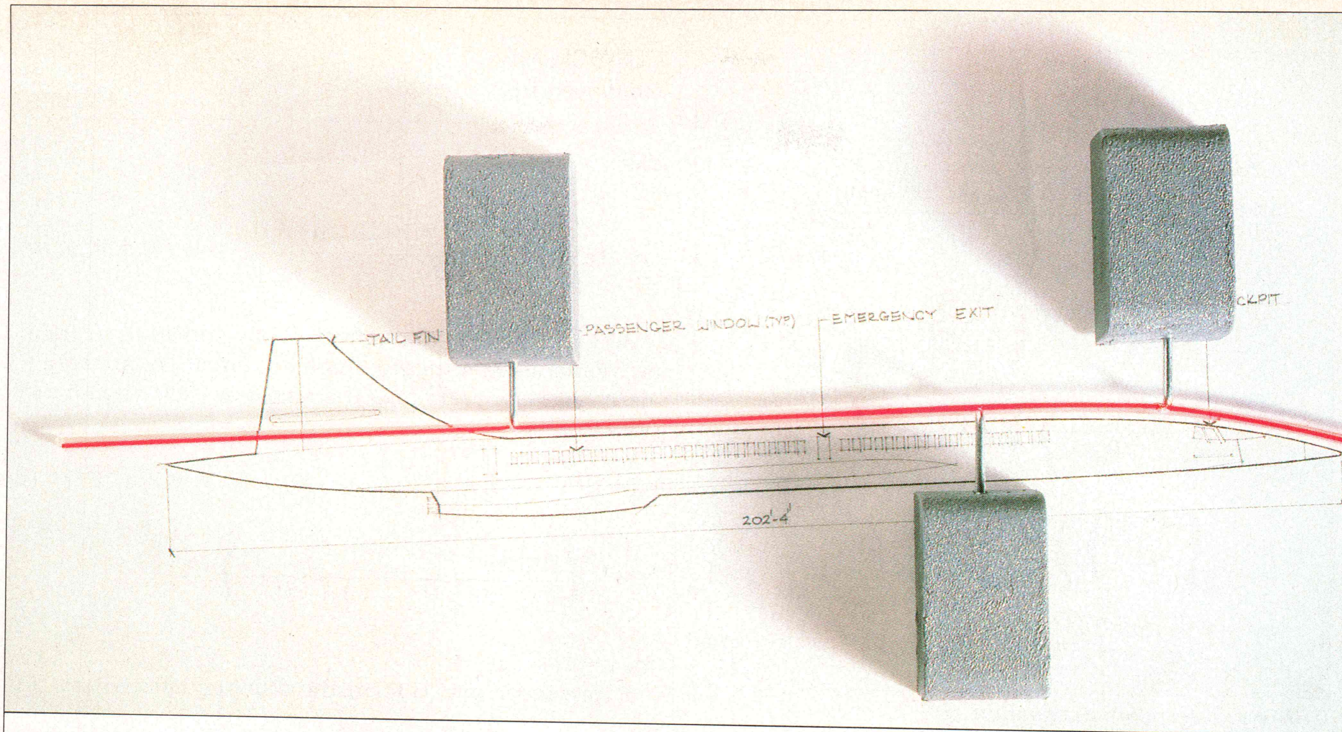
A Simple Explanation

Some of the math involved in spline calculations may be a bit daunting if you haven't had training in calculus and matrix mathematics. For those who want to get the gist of it without getting tangled up in equations, here's a short summary.

Because a spline is really nothing more than the graphs of a set of contiguous (endpoint-adjacent) cubic equations, all you need to know to draw it are the parameters of each of the equations. Naturally, there will be one such equation for each segment of the spline.

Because you know that the endpoints of each segment will coincide with the endpoints of the adjacent ones, and you know that the slopes of both curves will be the same at the joining points, it is possible to derive the equations

byHeart Software, 620 Ballantree Rd., West Vancouver, BC V7S 1W3, Canada



of the segment curves because, for any set of two points and two slopes, there is one and only one cubic curve segment that passes through the two points with the given slopes. Because each segment of the spline shares the same slope at its endpoints as its neighbors, you know that its first derivative (slope) will be the same at that point. Because the curves are the graphs of cubic polynomials, you know their second derivatives (change in slope) will be straight lines. Because (from the definition of the spline curve) the slope changes smoothly over the length of the curve, you know that the graph of the second derivatives of the spline equations will consist of a series of straight lines joined together at their endpoints.

A Rigorous Explanation

Beginning in this section, the math will get quite involved. If you're not into heavy math, you might want to skip down to the sections on the actual algorithms or you might want to skim through the math sections. It's not necessary to have a full understanding of the math in order to make the spline program work properly, but it helps.

Starting with a set of data points (the knots) stated as ordered horizontal coordinates x[i] (i = 1, . . . ,n) and corresponding vertical coordinates y[i], define the curve-to-be as the composite function:

$$y(x) = f[i](x) \text{ for } x[i] <= x < x[i+1];\ i = 1, \ldots, n-2; \quad \text{and } x[n-1] <= x <= x[n]$$

where each function f[i](x) is a cubic polynomial of the form:

$$f[i](x) = ax^3 + bx^2 + cx + d$$

where a, b, c, and d are constants. In other words, y(x) is really a set of functions, each of which is defined over an interval between two adjacent knots at (x[i], y[i]) and (x[i+1], y[i+1]).

Furthermore, let's define y'[i] and y''[i] as the first and second derivatives of y(x) at x = x[i]. Knowing that the set of functions f[i](x) must join at their endpoints (the knots of the spline) and also that their first and second derivatives are continuous at these points, you have the following continuity conditions:

$$\begin{array}{ll}
f[i](x[i]) = y[i] & i = 1, \ldots, n-1 \\
f[i-1](x[i]) = y[i] & i = 2, \ldots, n \\
f'[i-1](x[i]) = f'[i](x[i]) & i = 2, \ldots, n-1 \\
f''[i-1](x[i]) = f''[i](x[i]) & i = 2, \ldots, n-1
\end{array}$$

Because each function f[i](x) is a cubic polynomial, it follows that its second derivative is a linear function (a straight line) between its endpoints. If you define:

$$h[i] = x[i+1] - x[i]$$

then linear interpolation gives you:

$$f''[i](x) = \frac{y''[i] * (x[i+1] - x) + y''[i+1] * (x - x[i])}{h[i]}$$

Integrating this equation twice and selecting the constants of integration such that the continuity conditions are satisfied, you can derive the interpolation equation shown in Table 1, page 26.

Remember this equation—you'll use it later to interpolate the curve defined by y(x) between the given knots. But first you need to calculate the unknown coefficients y''[i] for all i between 1 and n.

Differentiating and evaluating the interpolation equation for x[i] yields:

$$f'[i](x[i]) = \frac{y[i+1] - y[i]}{h[i]} - \frac{h[i]}{6} * (2 * y''[i] + y''[i+1])$$

$$f[i](x) = \frac{y[i] * (x[i+1] - x) + y[i+1] * (x - x[i])}{h[i]} - \frac{h[i]^2}{6} * \left(y''[i] * \left(\frac{(x[i+1]-x)}{h[i]} - \left(\frac{x[i+1]-x}{h[i]} \right)^3 \right) + y''[i+1] * \left(\frac{(x-x[i])}{h[i]} - \left(\frac{x-x[i]}{h[i]} \right)^3 \right) \right)$$

for i = 1, . . . ,n

Table 1: *Interpolation equation*

CUBIC SPLINES
(continued from page 25)

and

$$f'[i-1](x[i]) = \frac{y[i] - y[i-1]}{h[i-1]} + \frac{h[i-1]}{6} * (y''[i-1] + 2 * y''[i])$$

Because the first derivatives of the functions at their endpoints are continuous, these two equations are equivalent. You can rearrange the terms of their right-hand sides to get:

$$h[i-1] * y''[i-1] + 2 * (h[i-1] + h[i]) * y''[i] + h[i] * y''[i+1] = 6 * \left(\frac{y[i+1] - y[i]}{h[i]} - \frac{y[i] - y[i-1]}{h[i-1]} \right)$$

for i = 2, . . . ,n−1.

Expressed in matrix form, the above equations show an

$$\begin{bmatrix} h[1] & c[2] & h[2] & 0 & 0 & 0 \\ 0 & h[2] & c[3] & h[3] & 0 & 0 \\ 0 & 0 & h[3] & c[4] & h[4] & 0 \\ 0 & 0 & 0 & h[4] & c[5] & h[5] \end{bmatrix} \begin{bmatrix} y''[1] \\ y''[2] \\ y''[3] \\ y''[4] \\ y''[5] \\ y''[6] \end{bmatrix} = \begin{bmatrix} d[2] \\ d[3] \\ d[4] \\ d[5] \end{bmatrix}$$

where

$$c[i] = 2 * (h[i-1] + h[i])$$

$$d[i] = 6 * \left(\frac{y[i+1] - y[i]}{h[i]} - \frac{y[i] - y[i-1]}{h[i-1]} \right)$$

Table 2: *f'[i](X [i]) and f'[i−1](X [i]) expressed in matrix form for n=6*

$$\begin{bmatrix} c[2] & h[2] & 0 & .... & 0 & 0 \\ h[2] & c[3] & h[3] & .... & 0 & 0 \\ 0 & h[3] & c[4] & .... & 0 & 0 \\ .... & .... & .... & .... & .... & .... \\ 0 & 0 & .... & h[n-3] & c[n-2] & h[n-2] \\ 0 & 0 & .... & 0 & h[n-2] & c[n-1] \end{bmatrix} \begin{bmatrix} y''[2] \\ y''[3] \\ y''[4] \\ .... \\ y''[n-2] \\ y''[n-1] \end{bmatrix} = \begin{bmatrix} d[2] \\ d[3] \\ d[4] \\ .... \\ d[n-2] \\ d[n-1] \end{bmatrix}$$

where

$$c[2] = (2 + j) * h[1] + 2 * h[2]$$
$$c[i] = 2 * (h[i-1] + h[i]) \quad \text{for } i = 3, \ldots, n-2$$
$$c[n-1] = 2 * h[n-2] + (2 + k) * h[n-1]$$

$$d[i] = 6 * \left(\frac{y[i+1] - y[i]}{h[i]} - \frac{y[i] - y[i-1]}{h[i-1]} \right) \quad \text{for } i = 2, \ldots, n-1$$

Table 3: *Matrix form of nonperiodic spline function*

interesting diagonal symmetry that you can take good advantage of later. Using n = 6 as an example, they look like those shown in Table 2, page 26.

A Variety of End Conditions

So far, you have n unknowns y''[i] but only n−2 conditions as expressed by the above equations. Two more conditions are required to obtain a unique solution for your curve y(x). Several variations are possible; I'll look at two of the more useful ones here.

The first is to specify that:

y''[1] = j * y''[2]

and

y''[n] = k * y''[n−1]

where j and k are arbitrary constants. With a bit of matrix manipulation, you get the equations shown in Table 3, page 26.

If the values of j and k are zero, you have y''[1] = y''[n] = 0. This is equivalent to a spline whose ends are not constrained beyond the end knots and is known as the "natural" cubic spline. A nonzero value for j or k is equivalent to bending an end of the draftsman's spline and will affect all of the interior cubic polynomial functions. The effect on the interior polynomials, however, rapidly decreases as you move away from the endpoints.

For some sets of knots, a nonzero value of j or k will result in a smoother interpolating curve at its corresponding end. A value of 0.5 is often appropriate. Be forewarned, however, that for some negative values the curve will be discontinuous. As it approaches these values, the end of the curve begins to oscillate, the peaks becoming larger and larger until they reach infinity at the exact values.

The above set of linear equations can be solved using Gaussian elimination. You must be careful, however. In

```
/* Reduce matrix to upper triangular form */

for i = 2 to i = n−2
    begin
        c[i+1] = c[i+1] − h[i] * h[i]/c[i]
        d[i+1] = d[i+1] − d[i] * h[i]/c[i]
    end

/* Solve using back substitution */

y''[n−1] = d[n−1]/c[n−1]
for i = n−2 to i = 2
    begin
        y''[i] = (d[i] − h[i] * y''[i+1])/c[i]
    end

y''[1] = j * y''[2]   /* End conditions */
y''[n] = k * y''[n−1]
```

Table 4: *Algorithm 1: Nonperiodic spline coefficient determination*

(continued from page 27)

its most general form, this method can require prodigious amounts of memory and millions of floating-point computations. Given 1,000 unknowns, Gaussian elimination needs storage for more than 1 million floating-point numbers and performs some 334 million multiplications and divisions! The loss of accuracy because of so many calculations can render the results meaningless.

Fortunately, the coefficient matrix described here is very sparse and symmetrical. The nonzero elements can be stored in a few linear arrays and the remainder ignored. By observing how Gaussian elimination solves the equations, you can modify the method to eliminate operations involving multiplication by and addition of zero. The result is Algorithm 1 (Table 4, page 27), which has very reasonable memory requirements and execution times—a cubic spline problem with 1,000 knots can be solved quickly on most personal computers, even those with less than 64K of memory!

In practice, array y''[ ] would be used initially to store the elements of array d[ ]. Then, as the elements of y''[ ] are solved during back substitution, they overlay the values of d[ ]. To implement this space-saving technique in Algorithm 1, change every instance of d[ ] to y''[ ].

The second variation is more interesting and comes from the need to interpolate data extracted from periodic phenomena. If you plot any periodic data in polar coordinates, a smooth curve between them forms a closed curve, with the endpoints of the curve meeting. Plotting the same data in rectilinear coordinates with the horizontal coordinates expressed over 360 degrees, it's easy to see that you can model the curve with a cubic spline function.

Because the curve is periodic, the endpoint vertical coordinates are by definition equal. In other words, y[1] = y[n]. You need to specify the end conditions such that the first and second derivatives of the curve are continuous with respect to each other at these points. Stated in mathematical terms, y'[1] = y'[n] and y''[1] = y''[n].

The second derivatives are easy—they can be expressed directly in matrix form. To use the first derivatives of y(x) at the endpoints, you need an equation that relates them to y(x) and its second derivative. Going back to the derivations for f'[i](x[i]) and f'[i−1](x[i]) and evaluating them for x[1] and x[n] respectively, you have:

$$f'[1](x[1]) = \frac{y[2] - y[1]}{h[1]} - \frac{h[1]}{6} * (2 * y''[i] + y''[2])$$

and

$$f'[n-1](x[n]) = \frac{y[n] - y[n-1]}{h[n-1]} + \frac{h[n-1]}{6} * (y''[n-1] + 2 * y''[n])$$

But y'[1] = y'[n], so:

$$6 * \left(\frac{y[2] - y[1]}{h[1]} - \frac{y[n] - y[n-1]}{h[n-1]} \right) =$$

$$h[n-1] * (y''[n-1] + 2 * y''[n]) + h[1] * (2 * y''[1] + y''[2])$$

Again with some matrix manipulation, you get the equations shown in Table 5, below. These equations can be solved efficiently and quickly with another modified version of Gaussian elimination, as shown in Table 6, page 30.

Other end conditions are possible. You can, for example, specify the slope of the spline at its endpoints by specifying the first derivatives at y[1] and y[n]. You can

$$\begin{bmatrix} c[2] & h[2] & 0 & \cdots & 0 & h[1] \\ h[2] & c[3] & h[3] & \cdots & 0 & 0 \\ 0 & h[3] & c[4] & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & h[n-2] & c[n-1] & h[n-1] \\ h[1] & 0 & \cdots & 0 & h[n-1] & c[n] \end{bmatrix} \begin{bmatrix} y''[2] \\ y''[3] \\ y''[4] \\ \cdots \\ y''[n-1] \\ y''[n] \end{bmatrix} = \begin{bmatrix} d[2] \\ d[3] \\ d[4] \\ \cdots \\ d[n-1] \\ d[n] \end{bmatrix}$$

where

$$c[i] = 2 * (h[i-1] + h[i]) \quad \text{for } i = 2, \ldots, n-1$$
$$c[n] = 2 * (h[1] + h[n-1])$$

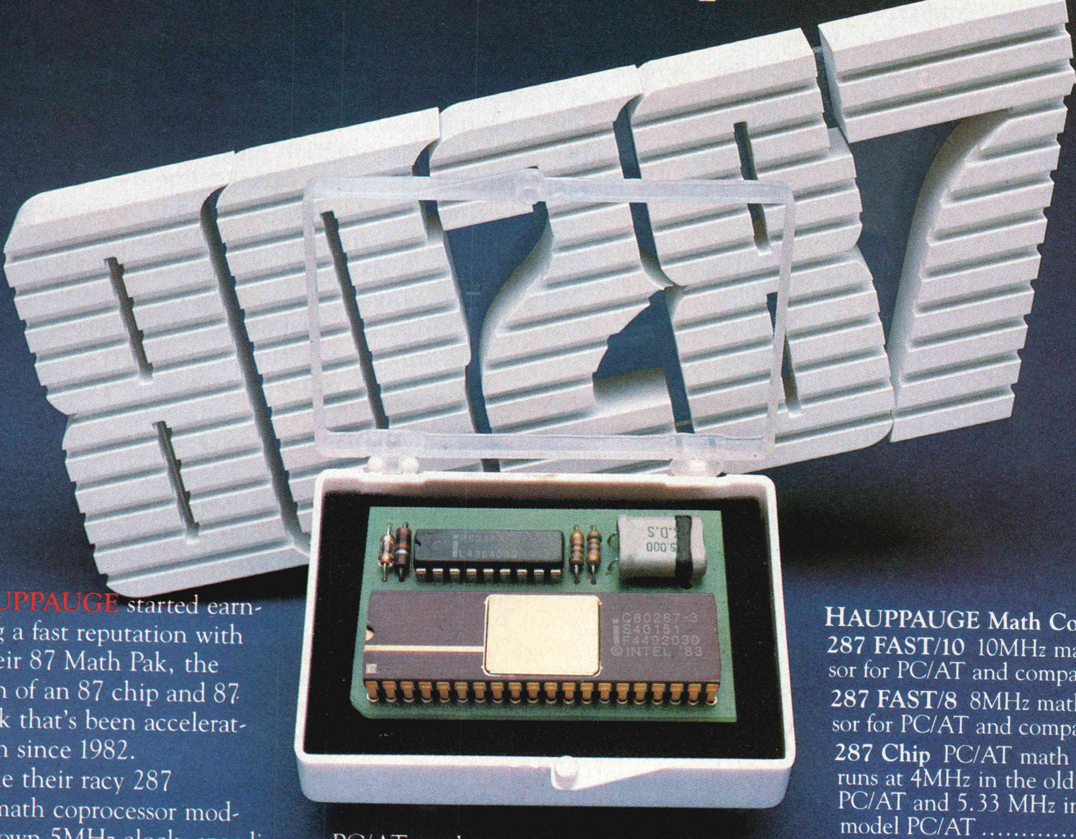
$$d[i] = 6 * \left(\frac{y[i+1] - y[i]}{h[i]} - \frac{y[i] - y[i-1]}{h[i-1]} \right) \quad \text{for } i = 2, \ldots, n-1$$

$$d[n] = 6 * \left(\frac{y[2] - y[1]}{h[1]} - \frac{y[n] - y[n-1]}{h[n-1]} \right)$$

Table 5: *Matrix form of periodic spline function*

HAUPPAUGE

Is Getting A Fast Reputation.



HAUPPAUGE started earning a fast reputation with their 87 Math Pak, the combination of an 87 chip and 87 Software Pak that's been accelerating PC math since 1982.

Next came their racy 287 FAST/5, a math coprocessor module with its own 5MHz clock, speeding up PC/AT math by 25%. (Pictured above.)

Now, Hauppauge Unveils the 287 FAST/10...
Our newest math coprocessor for the PC/AT, the 287 FAST/10 moves out at 8MHz—more than doubling the speed of each floating point math operation. The FAST/10 accelerates AutoCad, 1-2-3, Symphony, Turbo Pascal, Framework and more. The FAST/10 also runs in PC/AT compatibles including the Compaq Deskpro 286, Sperry PC/IT and most 286 accelerator boards.

...And the 87 Software Pak Version 6.0
Designed to steal the heart of programmers, the 87 Software Pak supports IBM's BASIC Compiler 1.0 and 2.0, and Microsoft's QuickBASIC, executing math-intensive programs up to 20 times faster! The 87 Software Pak also performs FFT's and Matrix operations. For example, a PC (or PC/XT) with an 87 Chip and 87 Software Pak can perform a 512-point complex FFT in just 1.1 seconds. What's more, a PC/AT with a FAST/10 inverts a 25 by 25 element matrix in under 1 second.

Circle **no. 274** on reader service card.

HAUPPAUGE Math Coprocessors

287 FAST/10 10MHz math coprocessor for PC/AT and compatibles.....**$469**

287 FAST/8 8MHz math coprocessor for PC/AT and compatibles **$379**

287 Chip PC/AT math coprocessor—runs at 4MHz in the old model IBM PC/AT and 5.33 MHz in the new model PC/AT **$219**

87 Chip Math coprocessor for IBM PC, PC/XT and compatibles...**$129**

87-2 Chip Math coprocessor for 8MHz PC compatibles...**$195**

HAUPPAUGE Math Coprocessor Paks

87 Math Pak V.6.0 87 chip and math coprocessor software support for IBM BASIC Compiler 1.0, 2.0 and Microsoft's QuickBASIC. Plus, Matrix and FFT support, one year of free updates, complete source code and "8087 Applications and Programming" **$279**

87 Software Pak V.6.0 Math coprocessor software support as in the 87 Math Pak, but without 87 chip.................**$180**
With any Hauppauge math coprocessor **$150**

Recalc + Math coprocessor support for 1-2-3 version 1A ..**$ 95**
With any Hauppauge math coprocessor ....**$ 49**

HFT + Complete Hayes Fourier Transform Package **$125**
With any Hauppauge math coprocessor ....**$ 79**

The 287 FAST/8 Doubles Your PC/AT's Math Speed!
Help your PC/AT get a fast reputation with Hauppauge's new 287 FAST/8. Call today, or contact your local computer dealer to learn more about Hauppauge's racy product line. And ask for "87 Q & A," our free booklet on math coprocessors.

Hauppauge Computer Works, Inc.
358 Veterans Memorial Highway, Suite MSI,
Commack, New York, USA 11725
516-360-3827 • TELEX: 262939-HCW

Hauppauge
(Pronounced "Ha-pog")

We acknowledge the following registered trademarks: 1-2-3 and Symphony: Lotus Development Corporation; AutoCad: AutoDesk, Inc.; IBM: PC, PC/AT, PC/XT; Compaq: Deskpro; Sperry: PC/IT; Texas Instruments: Business Pro; Ashton-Tate: Framework.

CUBIC SPLINES
(continued from page 28)

also specify a linear combination of first and second derivatives at the endpoints. The two examples presented here, however, will generally prove the most useful for interpolative curve fitting.

Specifying the end conditions and solving the appropriate set of linear equations gives you the coefficients you need to solve your interpolation equation. (Note that this equation remains the same no matter what end conditions have been specified.) For any given value of x within the range of values spanned by the knots, you need only determine the two knots between which the value lies. This gives you the value of i to insert in the interpolation equation and with it the appropriate coefficients y"[i] and y"[i+1] to use in solving for the corresponding y coordinate.

What about the related problem of fitting a smooth surface to data plotted in three dimensions? If the data is regularly spaced in two of those dimensions (say the x−y plane), you can calculate a family of curves in parallel x−z planes. Each curve is the intersection of the x−z plane with the surface. Then, for any perpendicular y−z plane, your knots are the intersection of the x−z plane curves with the y−z plane. From these, you can calculate the intersection of your surface with the y−z plane. With this method, you can determine any point on the surface uniquely.

```
/* Initialize array e[ ] as nth column of matrix M[ ][ ] */

e[2] = h[1]
for i = 3 to i = n-2
    begin
        e[i] = 0
    end
e[n-1] = h[n-1]
e[n] = c[n]

/* Initialize variable f as matrix element M[n][1] */

f = h[1]

/* Reduce matrix to upper triangular form */

for i = 2 to i = n-2
    begin
        c[i+1] = c[i+1] - h[i] * h[i]/c[i]
        d[i+1] = d[i+1] - d[i] * h[i]/c[i]
        e[i+1] = e[i+1] - e[i] * h[i]/c[i]
        d[n] = d[n] - d[i] * f/c[i]
        e[n] = e[n] - e[i] * f/c[i]
        f = -f * h[i]/c[i]   /* Now matrix element M[n][i] */
    end
    f = f + h[n-1]   /* Now matrix element M[n][n-1] */
    d[n] = d[n] - d[n-1] * f/c[n-1]
    e[n] = e[n] - e[n-1] * f/c[n-1]

/* Solve using back substitution */

y"[n] = d[n]/e[n]
y"[n-1] = (d[n-1] - e[n-1] * y"[n])/c[n-1]
for i = n-2 to i = 2
    begin
        y"[i] = (d[i] - h[i] * y"[i+1] - e[i] * y"[n])/c[i]
    end

y"[1] = y"[n]   /* End condition */
```

***Table 6:** Algorithm 2: Periodic spline coefficient determination*

Final Words

I could have demonstrated the above algorithms using a small BASIC program; however, the Unix operating system offers a utility called spline that is much more comprehensive. Heeding once again Richard Stallman's call ("The GNU Manifesto," *DDJ*, March 1985) for placing Unix in the public domain ("FGREP," *DDJ*, September 1985, was my previous response), the accompanying "demonstration" program (SPLINE.C) is a full emulation of the Unix spline utility. (See Listing One, page 72.)

If you would rather not spend an evening or two entering and (inevitably) debugging SPLINE.C, you can purchase machine-readable versions for $35 from byHeart Software, 620 Ballantree Rd., West Vancouver, BC V7S 1W3, Canada. Supported disk formats are CP/M 8-inch SSSD and MS-DOS (2.x) 5¼-inch DSDD. Included on the disk are the source code in C for SPLINE.C and FGREP.C, their executable programs, and the text from this article and "Parallel Pattern Matching and FGREP" (*DDJ*, September 1985).

Cubic splines are an elegant solution to the problem of fitting curves to a set of given points in an x−y plane. An understanding of the mathematics used to develop them is not essential. The simplicity and efficiency of the algorithms involved should encourage anyone interested in graphics or data analysis to add cubic splines to their software toolboxes.

Bibliography

Bell Telephone Laboratories. *UNIX Programmer's Manual*, vol. 1. New York: Holt, Rinehart & Winston, 1983. 145.

Forsythe, George; Malcolm, Michael; and Moler, Cleve. *Computer Methods for Mathematical Computations.* Englewood Cliffs, N.J.: Prentice-Hall, 1977. 70−83.

Hamming, R. W. *Numerical Methods for Scientists and Engineers.* 2d ed. New York: McGraw-Hill, 1973. 349−355.

Hildebrand, F. B. *Introduction to Numerical Analysis.* New York: McGraw-Hill, 1974. 478−494.

DDJ

(Listing begins on page 72.)

Vote for your favorite feature/article.
Circle Reader Service **No. 2.**

PC MAGAZINE EDITOR'S CHOICE

*...one package stands out as the best support available for Turbo Pascal programmers: **Blaise Computing's Turbo Power Tools.** This definitive set of prewritten Pascal functions and procedures will make the life of any programmer—from the beginner to the hard-core professional—easier and more productive.*

ANOTHER PLUS FROM BLAISE COMPUTING

Power Tools Plus™

Window Routines. Memory Resident Routines. Routinely.

The best just got better! Turbo POWER TOOLS, acclaimed as the best programmer support package for Turbo Pascal, now has even more functions, more detailed documentation and more sample programs.

NO SECRETS

Turbo POWER TOOLS PLUS is crafted so that the source is efficient, readable and easy to modify. We don't keep secrets! We tell you exactly how windows are managed, how interrupt service routines can be written in Turbo Pascal, and how to write memory resident programs that can even access the disk. Maybe you've heard of some undocumented DOS features that resident programs use to weave their magic. Turbo POWER TOOLS PLUS documents these features and lets you make your own magic!

Here's just part of the PLUS in Turbo POWER TOOLS PLUS:

- **WINDOWS** that are stackable, removable, with optional borders and a cursor memory.
- **FAST DIRECT VIDEO ACCESS** for efficiency.
- **SCREEN HANDLING** including multiple monitor and EGA 43-line support.
- **POP-UP MENUS** which are flexible, efficient and easy to use, giving your applications that polished look.
- **INTERRUPT SERVICE ROUTINES** that can be written in Turbo Pascal without the need for assembly language or inline code.
- **INTERVENTION CODE** lets you develop memory resident applications that can take full advantage of DOS capabilities. With simple procedure calls, you can "schedule" a Turbo Pascal procedure to execute either when a "hot key" is pressed, or at a specified time.
- **PROGRAM CONTROL ROUTINES** allow you to run other programs from Turbo Pascal, and even execute DOS commands.
- **MEMORY MANAGEMENT** allows you to monitor, allocate and free DOS-controlled memory.
- **DIRECTORY AND FILE HANDLING** support to let you take advantage of the newer features of DOS including networking.
- **STRING** procedures allowing powerful translation and conversion capabilities.
- **FULL SOURCE CODE** for all included routines, sample programs and utilities.
- **DOCUMENTATION, TECHNICAL SUPPORT** and attention to detail that have distinguished Blaise Computing over the years.

Turbo POWER TOOLS PLUS** supports Turbo Pascal Version 2.0 and later and is just **$99.95.

*Another quality product from Blaise Computing: **Turbo ASYNCH PLUS™***

A new package which provides the crucial core of hardware interrupt support needed to build applications that communicate. ASYNCH PLUS offers simultaneous buffered input and output to both COM ports at speeds up to 9600 baud. The XON/XOFF protocol is supported. Now it also includes the "XMODEM" file-transfer protocol and support for Hayes compatible modems.

The underlying functions of Turbo ASYNCH PLUS are carefully crafted in assembler for efficiency and drive the UART and programmable interrupt controller chips directly. These functions, installed as a runtime resident system, require just 3.2K bytes. The high level function are all written in Turbo Pascal in the same style and format as Turbo POWER TOOLS PLUS. All source code is included for just $99.95.

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

ORDER TOLL-FREE 800-227-8087
Calif. residents call (415) 540-5441

YES, send me the PLUS I need! Enclosed is $_____ for
☐ Turbo POWER TOOLS PLUS ☐ Turbo ASYNCH PLUS ☐ OTHER _____. (CA residents add 6½% Sales Tax. All domestic orders add $10.00 for Federal Express shipping.)

Name: _____ Phone: (___) _____
Shipping Address: _____
City: _____ State: _____ Zip: _____
VISA or MC #: _____ Exp. Date: _____

Circle **no. 159** on reader service card.

A General First-Order Sorting Algorithm

by Robert A. McIvor

This linear-time sort was once used on punched cards.

Most widely used sorting algorithms, such as the Shell sort and quicksort, require a sorting time approximately proportional to $n\log_2 n$, where n is the number of sort keys, whereas the bubble sort requires a time proportional to n^2. Often it is stated erroneously that it has been proved that no sorting algorithm can improve on an $n\log_2 n$ time. The proof refers only to algorithms based on exchanges, however, as Knuth states clearly in volume 3 of his series *The Art of Computer Programming*.

There is a sort algorithm called the radix sort that is not based on exchanges. This sort has been known and used since long before the time of electronic computers and was generally used for sorting punched cards, but it does not seem to have been widely adapted for computer use. The time required to do such a sort is proportional to n.

In seeking a reason for the slighting of this algorithm, which I have used successfully on a variety of computers for about 20 years, I have come to the following conclusion. There must be an intuitive feeling on the part of programmers that, whereas sorts requiring time k_1n will always be faster than those requiring $k_2n \log_2 n$ when the value of n is sufficiently great (where k_1 and k_2 are arbitrary constants), for practical values of n that fit into a computer's RAM, k_1 is sufficiently greater than k_2 to nullify the advantage. As I will demonstrate, this assumption is invalid in many cases.

Robert A. McIvor, 3100 Carling Ave., Apt. 920, Nepean, ON K2B 6J6, Canada

How the Radix Sort Works

The basis of the radix sort algorithm is to divide the sort keys into two lists, with their placement in a list being dependent on whether the least significant bit is set or not. The two lists are then concatenated with the list in which the bit is not set placed first. This step is repeated for each bit in each sort key, working from the least significant end to the most significant end. When this procedure is complete, the file is sorted.

The chief disadvantage of this method when compared to exchange sorts is that the time required for completion has the same dependence on the number of bytes in the key as it does for the number of keys—that is, it takes the same time to sort 10,000 1-byte records and 1,000 10-byte records. Exchange sorts, on the other hand, are much less dependent on key length than on the number of keys because it is usually unnecessary to compare every byte in two keys to determine which is the greater.

Another possible disadvantage of the algorithm as presented here (a sort algorithm for linked lists) is the additional space overhead required. Each key must have an additional 2–4 bytes allotted for a pointer address. Although this is perhaps unreasonable for 1-byte sort keys, the disadvantage becomes less and less significant with longer sort keys. Furthermore, if the data to be sorted is already in a linked list, no additional space is required. For sorts in which keys must be extracted from a record and manipulated to perform special sorts, such as sorting signed numbers, sorting in reverse order, or sorting in a special collation order, forming a linked list of the sort keys does not usually entail much additional effort.

The Test Programs

Listing One, page 86, is the radix sort as coded for the Macintosh in Aztec C. The include files are needed to provide the definitions of *Random( )* and *TickCount( )* used in the timing routine. The routine *MaxApplZone( )* provides space in the application heap for the sort keys. It must be called before anything else, and the *lmalloc* must be given a noncalculated number or the exit branch of insufficient space is taken. Also, the call seems to be required in the routine in which the allocation occurs. RAM disks and cache programs for the Macintosh may have adjusted the application heap, leaving no space for allocation. Allocation is necessary for record counts in the thousands because the space permitted for data declared in the program is limited. KEYSIZ (the size of the sort key) is declared at compile time to avoid editing for each change.

The sort program is passed the key size and the pointer to the first record. Two additional pointers (*first* and *last*) follow the two lists created at each pass, whose heads are given by the pointers *start* and *start2*. The pointer *temp* follows the combined list during each pass. Each bit from

Number of Records	Bytes in Key											
	1	2	3	4	5	6	7	8	9	10	11	12
1000	19	37	56	74	92	112	130	149	168	186	205	223
2000	37	74	111	148	185	223	260	297	335	372	409	443
3000	56	111	166	222	277	334	390	446	502	557	613	663
4000	74	148	221	295	369	446	520	594	670	743	818	885
5000	93	185	278	369	462	557	650	743	835	929	1022	1107
6000	112	222	333	443	554	669	780	892	1002	1115	1225	1328
7000	131	259	388	517	646	780	910	1040	1170	1301	1430	1549
8000	149	296	444	590	739	891	1040	1189	1336	1486	1634	1770
9000	167	333	500	665	831	1002	1170	1337	1504	1671	1838	1992
10000	186	370	554	739	923	1114	1300	1485	1672	1856	2042	2213

Times are in 60ths of a second.

***Table 1:** Execution times of radix sort on Macintosh*

Number of Records	Bytes in Key											
	1	2	3	4	5	6	7	8	9	10	11	12
1000	104	129	146	162	176	203	205	229	254	250	240	290
2000	256	321	344	389	446	509	502	515	599	584	617	685
3000	388	537	562	623	717	783	822	848	933	1034	069	095
4000	570	775	857	966	1136	1245	1185	1255	1475	1464	1571	1685
5000	778	957	1149	1228	1516	1554	1549	1630	1909	1929	2112	2148
6000	930	1240	333	1462	1788	1902	1936	1942	2205	2410	2552	2762
7000	1044	1547	1645	1774	2034	2287	2471	2437	2771	3062	3170	3295
8000	1347	930	2111	2429	2806	2859	2975	3087	3616	3625	3855	4240
9000	1511	2069	2212	2483	3008	3257	3057	3270	3856	4111	4482	4563
10000	1736	2400	2639	2899	3544	3648	3845	4270	4668	4695	4688	5216

***Table 2**: Execution times of Shell sort on Macintosh*

Key Count	Sort Key Length in Bytes									
	1	2	3	4	5	6	7	8	9	10
1000	26.7 5.4	30.3 10.8	33.2 16.2	34.6 21.6	37.2 27.0	39.8 32.4	42.5 37.8	46.0 43.2	47.5 48.6	48.1 54.0
2000	60.7 10.8	68.2 21.6	81.2 32.4	85.4 43.2	94.4 54.0	101.0 64.8	105.0 75.6	118.0 86.4	118.0 97.2	118.2 108.0
3000	92.1 16.2	132.0 32.4								198.0 162.0
4000	138.0 21.6	188.0 43.2								
5000	188.0 27.0	267.0 54.0								
6000	239.0 32.4									
7000	264.0 37.8									
8000	328.0 43.2									
9000	406.0 48.6									
10000	438.0 54.0									
11000	459.0 59.4 (calculated)									
12000	535.0 64.8 (calculated)									

Top value is for shell. Bottom value is for radix.

***Table 3:** Comparative sort time in seconds for Shell sort and radix sort on Z80*

SORTING ALGORITHM
(continued from page 32)

least to most significant is isolated in turn, and the byte is added to the *first* and *last* lists depending on whether it is 0 or 1. When the end of the *temp* list is reached, the *last* list is terminated with a 0 pointer, and the last member of the *first* list is pointed to the head of the *last* list (*start2*). When all bits have been traversed, the pointer *start*, which points to the head of the sorted list, is returned. In addition, checks are made for empty lists, and the appropriate action is taken.

I compared the radix sort algorithm times with those obtained from the Shell sort algorithm given on page 116 of *The C Programming Language* by Kernighan and Ritchie. I've provided a listing of my version of the Shell sort for comparison (Listing Two, page 87). The initialization, of course, is different for the Macintosh.

Some Test Results

In both the radix and the Shell sorts, a random function was used to provide data for sorting. The times for the radix sort, unlike those for the Shell sort, are data independent. Tables 1 and 2, page 33, show the sort times for the radix and Shell sorts on a 512K Macintosh.

I also programmed the algorithms in BDS C for a Z80 computer running at 2 MHz. In both cases the radix sort is superior for sort keys shorter than 12 bytes when the number of records exceeds 1,000. For 10,000 1-byte records, the radix sort is more than nine times faster. The crossover point for 12-byte keys occurs at approximately 500 records, and for keys of 7 bytes or less, the crossover is at 100 or fewer records. Table 3, page 33, shows the results for the Z80 system.

The execution time of the radix sort for the Z80 system can be expressed by the formula $(5.4 \times 10^{-3})mn$ seconds, where m is the number of bytes in the sort key and n the number of keys to be sorted. The Macintosh execution time was $(3.11 \times 10^{-4})mn$ seconds—more than 17 times faster. The ratio of times for the Shell sort was similar.

The radix sort algorithm was also coded in Z80 assembly language. The run times are approximately six times faster—the measured time for the radix sort was $(9.1 \times 10^{-4})mn$ seconds at 2 MHz.

You might think that by masking two bits at a time and dividing the keys into four lists, an additional time savings of about 50 percent would be achieved. In fact, the saving is only about 17 percent because of the increased overhead in the inner loop.

Bibliography

Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*. Englewood Cliffs, N.J.: Prentice-Hall, 1978.

Knuth, Donald. *The Art of Computer Programming, Volume 3. Sorting and Searching*. Reading, Mass.: Addison-Wesley, 1969.

DDJ

(Listings begin on page 86.)

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**

NEW!! NEW!!

TURBO SCREEN/APPLICATION GENERATOR

Give your application a Turbo Boost with the Most Advanced Turbo Pascal Environment Available. Have Turbo Master generate your screen, file handling and menu Programs.

For $99.95 receive 6 Floppy Disks & Manual.

TURBO ISAM MASTER

Generates 'Ready to Run' Turbo Programs Using BORLAND'S Turbo Pascals Database TOOLBOX or optional SOFTCRAFT'S BTRIEVE.

With a few key strokes you can generate the following Pascal programs/includes files. Automatically interfaces to Turbo Screen Master.

- **MASTER DATABASE PROGRAM** - Generates Pascal Program Code for the following functions
  - —**Add a Record** - Allows both duplicate & unique only keys. Change the up to 48 fields comprising the primary key and the up to 7 secondary keys at any time during input, and the validity of the keys are checked and the ISAM key files are automatically adjusted.
  - —**Delete a Record** - Shows the record to be deleted on the screen and allows you to change your mind about deleting, and then adjusts the keys files automatically for the up to 8 keys. The disk space is reused automatically by the programs.
  - —**Edit a Record** - with a key change allowed.
  - —**Search Database by Key** - The ability to display the keys on the screen or printer in sorted order.
- **DATA BASE RECOVERY PROGRAM.** This program recovers your database if it corrupted by a power outage or certain hardware failures.
- **MULTIPLE ISAM FILES** - Can be used at the same time in a single program. The generator also produces context sensitive instructions on how to integrate the generated program into your main program.
- **DOCUMENTATION** - Print screens and ISAM specifications. Also inline program documentation is generated.

TURBO MENU MASTER

GENERATES 'Ready to Run' Turbo Programs with Dedicated Screens or Windows.

- **FEATURES:** Has Menu Data Base so it's easy to modify menus.

PROVIDES SELECTION BY:
—Press of a number
—Press of a function key
—Press the high lighted letter
—Use the arrow keys

ADAPTIVE SCREEN COLORS — Different screen colors are automatically used for color or monochrome display. This allows you to provide a better interface for the user.

INTERACTIVE MENU BUILDER
—Allows for the automatic entry and reorder of selections
—Offers easy color selection
—Allows for choice of procedure, chain, execute or comment code generation for each selection
—Has startup Menu that YOU design.
—Provides for the integrated Display and control of the Key Lock Status.

TURBO RESIDENT SCREEN CAPTURE

which allows you to capture Text screens from any running program.

TURBO SCREEN MASTER

GENERATES 'Ready to Run' Mulit-screen programs with advanced field definition.

- **USER SPECIFIED PROCEDURES** can be called before and after data entry for each field. This allows **SCREEN GENERATION THAT IS INDEPENDENT OF USER INSERTED PROGRAM STATEMENTS.**
- **YOU CAN CUSTOM PROGRAM** special field edits and processes which Screen Master automatically includes into the screen program. This allows for "On the Fly **FUNCTION KEY AND WINDOW ROUTINES.**
- **RECORD FORMATS** and initialization routines for Structures and Arrays are automatically generated.
- **CAN SPECIFY** a Protected Field to be automatically redisplayed if its value changes.
- **TYPES INCLUDE** Strings, Yes/No, Time, Male/Female and Numbers.
- **FIELDS CAN BE STORED** (declared) as Boolean, Integer, Real, Character, or String.

COMPARISON WITH OTHER PRODUCTS	TURBO SCREEN MASTER	SCREEN SCULPTURE Ver - 1.01
Full support for structures, arrays and Declarations specifications	YES	NO
Full support for user written procedures, function keys & help screens	YES	NO
Etch-A-Sketch Border Drawings	YES	NO
Point and Paint color interaction	YES	YES
Border color control	YES	NO
User defined valid character sets	YES	NO
Display of Caps/Num Lock Status	YES	NO
Optional realtime initialization of data/time	YES	NO

TURBO RESIDENT ISAM

Replaces Borland's Toolbox ISAM method

- Eliminates 8k of code space that Borland's Turbo Toolbox ISAM requires in your program.
- Puts the Key Buffer Data Area outside of your program's data space.
- Speeds Compling
- Eliminates errors caused by inconsistant ISAM specifications between ISAM files.
- Increases program loading time.
- Reduces object program size.
- Offers SUPER-FAST record creation and lookup. For example, it can CREATE over 20 single key records PER SECOND on an IBM AT running at a 9 meg clock rate.

OUR USERS REPORT

- *"Since Fall of 85, I have generated over 300 program modules with it and find it to be just what I needed. Most of all the modules represent 5000 to 8000 lines of Pascal Code"* Oner Systems
- *"By being able to produce a 21 screen and menu control demo so quickly helped me obtain the contract."*
- *"Speeded up my screen development by 6 times"* Elexor Associates.
- *"Has many of the features of the Super Mini development tools costing $10,000."* Applied Micro Systems.
- *"We developed 3 Vertical Market Applications in the 6 months we had your system."* Absolute Systems.

★★★★★★★★★★★★★★★★★★★★★

BTRIEVE INTERFACE MODULE

- Allows full multiuser record locking and
- Automatic file recovery for the industry's most popular LANs. Works with the industry's leader of professional databases for multiuser LAN's (optional). **$99.95**
- Requires Btrieve by SoftCraft, Inc.

★★★★★★★★★★★★★★★★★★★★★

TURBO MASTER TOOLS

The Turbo Toolkit Master includes procedures for full control over all screen attributes, advanced string functions, automatic control of multiple help screens, saving and retrieving screens to RAM buffers, Caps/Num/Scroll/Lock control procedures, and report procedures.

RISK FREE TRIAL

Try the demo package included for 30 days. If not pleased return for a full refund.

Credit Card Orders Call:

1-800-821-9503

In Florida (800) 342-0137

For Other Information Call (305) 892-5686

NO ROYALTIES on Generated Programs

YES, ENCLOSED IS ____________

- ☐ **Turbo Master Tools**
- ☐ **Turbo Menu Master**
- ☐ **Turbo Screen Master**
- ☐ **Turbo ISAM Master**
- ☐ **Turbo Screen Capture**
- ☐ **Resident ISAM**

$99⁹⁵

Systems Requires: IBM PC,XT,AT or 100% Compatible -246K MS DOS 2.0 or Higher, Turbo Pascal 3.0 - 2 DD/DS Disk Drives.

US orders add $7.50 S&H

All foreign orders add $15 per product ordered.

Name: ______________ **Phone:** ______________

Shipping Address: ______________

City: ______________

State: ______________

Zip: ______________

VISA or MC #: ______________

Exp. Date ______________

C.O.D.'s will be accepted. Outside USA: make payment by bank draft, payable in US dollars drawn on a US bank.

HAWAIIAN VILLAGE COMPUTER SOFTWARE
1109 Pennsylvania Ave., St. Cloud, Florida 32769

Circle no. **233** on reader service card.

Turbo Prolog: The Language

by Michael Swaine

The cut operator is the feature PROLOG most needs to lose.

The story was too good to be true. I had to believe it.

A software developer of my acquaintance had designed an AI product and was searching for the best development environment for its implementation. Having worked with *PROLOG*, he knew he wanted to use it, if he could only find an implementation that met his needs. Borland, he heard, had acquired a fast *PROLOG* compiler—not an interpreter, but a compiler—and would be selling it for less than $100. Even though price was not his biggest concern, he felt he owed it to prudence to check this out. He called Borland.

Because he had a product in the works, the developer's questions were specific and technical, just as, because Borland had not yet released the product and had not developed it in-house, the answers he got initially were vague and unsatisfying. Persevering, he finally penetrated deep within the company to one programmer who really knew the product, the in-house expert. Although the programmer was knowledgeable and forthright, some of his answers surprised the developer, who consequently posed even more probing questions about the features and capabilities of the implementation, eliciting even more—to the developer—surprising answers. Finally, the programmer blurted out, "Well, you know it's not PROLOG; it's Turbo Prolog."

Apocryphal, no doubt. One of the goals of this review will be to decide what poetic truth there may be in the story.

Michael Swaine, 501 Galveston Dr., Redwood City, CA 94063.

"Core" PROLOG

PROLOG is a declarative as opposed to an imperative language, meaning that a program consists of a set of statements of fact rather than of a list of instructions. The language itself has the ability to do some of what in other languages is the programmer's job because PROLOG embodies an inference engine based on the techniques of resolution and unification. Resolution as a language basis is known to be logically complete in the sense that it can generate any of the logical implications of the facts and rules in a knowledge base. The imperative mode, the flow of control, and the logical inference process in a PROLOG program are all handled on an application-independent basis by the interpreter—or, now that PROLOG compilers are coming into being, at any rate not by the programmer. In principle, the programmer simply throws statements at PROLOG, and PROLOG deduces what is deducible (prompted by questions, referred to in PROLOG as goals).

If the above description makes the programmer's job sound too simple, note that the statements can be contingencies on variables, such as these two statements:

```
grandfather(X,G) :-
        father(P,G), mother(X,P).
grandfather(X,G) :-
        father(P,G), father(X,P).
```

which state that the grandfather relationship holds between entities *X* and *G* if the father relationship holds between *P* and *G* and the mother relationship holds between *X* and *P*, or if the father relationship holds between *P* and *G* and the father relationship also holds between *X* and *P*. These rules, together with a database of facts such as the following about parents and their children:

```
mother(john, rita).
father(jacob, eli).
father(rita, luigi).
father(john, jacob).
```

permit automatic deduction of implications, as in the following dialogue:

```
you:      grandfather(john,G)
program:  G = eli
program:  G = luigi
```

in which you ask for and receive the names of John's grandparents.

Despite the curious fact that the order of statements (for example, the order of the four statements about parenthood above) is often irrelevant to the successful execution of PROLOG programs, it is quite relevant to their efficient execution, and a PROLOG programmer has to expend some effort structuring programs for efficiency.

For a solid presentation of PROLOG, you should read *Programming in Prolog*, second edition, by W. F. Clocksin and C. F. Mellish. A good introduction for programmers is Dave Cortesi's "Tour of PROLOG" (*DDJ*, March 1985). Details of these and other sources are

listed at the end of this review.

What you won't find anywhere is the official definition of the language. Not only is there no such thing as standard PROLOG, but it's also not even clear that PROLOG is a language by everyone's definition. The closest thing we have to an official, broadly accepted definition is Clocksin and Mellish's "core" PROLOG. Clocksin and Mellish, though, are more descriptive than prescriptive regarding linguistic diversity and even seem to acknowledge that PROLOG may be less important as a language in itself than for the more powerful languages that will be developed in it or from it. PROLOG, they say in the preface to the second edition of their book, "is now seen as a potential basis for an important new generation of programming languages and systems."

PROLOG as it now stands has some deficiencies, and we have reason to look forward to that new generation of programming languages. For one thing, one of the few mechanisms for limiting explicitly the search space for solutions is the fairly awkward and implicit technique of ordering clauses and conjuncts. Another mechanism is the *cut* operator, which Clocksin and Mellish identify as the feature PROLOG most needs to lose.

Then there are the limitations of resolution, as discussed by Michael R. Genesereth and Matthew L. Ginsberg ("Logic Programming," *Communications of the ACM*, September 1985). A good logic programming system, they point out, needs to be able to "draw conclusions from uncertain data, reason analogically, and generalize its knowledge appropriately." That's not resolution, and it's not PROLOG.

What is PROLOG, officially? On page 111 of their book, Clocksin and Mellish erect a "core" PROLOG, built of the features most often found in existing implementations, on top of "pure" PROLOG; the built-in predicates many people regard as an integral part of the language proper are in fact part of the patchwork core, not the pure essence. The body of PROLOG explicitly put forth in Clocksin and Mellish is not a full language, and implementers have extended it as they saw fit. Finally, although Clocksin and Mellish have nevertheless cobbled together some sort of average implementation in their core PROLOG, a separate strand of development is represented by micro-PROLOG, a version with a radically different syntax, described in *micro-PROLOG: Programming in Logic* by K. L. Clark and F. G. McCabe.

Despite the limitations of resolution and the lack of definition of the PROLOG language, powerful and efficient implementations of PROLOG have been developed in Europe, Australia, Japan, and the United States. Several PC-based PROLOGs now exist (see the box below), one of the most interesting of which, at least in terms of claims being made about its speed, is Borland's Turbo Prolog. Judging by the discussions on *DDJ*'s CompuServe forum, there is much difference of opinion about the product.

Because there is no clear standard against which to weigh Turbo Prolog, and because vendors of other personal-computer-based PROLOG implementations seem to have had something else in mind when they developed their products, it makes sense to look at Turbo Prolog in isolation; *DDJ* does, however, plan to follow up on this review with a comparative review of PROLOG implementations. This review focuses on Borland's Turbo Prolog from several directions: as an implementation of PROLOG, using Clocksin and Mellish core PROLOG as a soft standard; as a language and software development environment in its own right; as a learning environment; and as a phenomenon in the world of software tools. On the way, it addresses some of the claims being made about Turbo Prolog.

Claim 1: Not Full PROLOG

Turbo Prolog is not full PROLOG.

You hear this claim often, most often from PROLOG purists and Borland competitors. It's certainly true, even by a fairly flexible definition of the language, that Turbo Prolog lacks some of the requisite elements of a full PROLOG implementation. The left column of Table 1, page 38, shows the major features of core PROLOG that are missing in Turbo Prolog. Many syntactical differences are left out of the table; Turbo Prolog largely ignores core PROLOG *I/O*, substituting its own rich collection of *I/O* primitives, screen-handling functions, and graphics commands. The Borland syntax is eclectic.

Turbo Prolog does, however, provide the bulk of what makes up core PROLOG. (Henceforth I'll drop the word *core*, but keep in mind that I

MS-DOS PROLOG Implementations

Ada PROLOG
Automated Design Associates
1570 Arran Way
Dresher, PA 19025
(215) 646-4894

Arity/PROLOG
Arity Corp.
336 Baker Ave.
Concord, MA 07142

Turbo Prolog
Borland International
4585 Scotts Valley Dr.
Scotts Valley, CA 95066
(408) 438-8400

PROLOG/i and PROLOG/m
Chalcedony Software
5580 La Jolla Blvd., #126B
La Jolla, CA 92037
(619) 483-8513

PROLOG1 and PROLOG2
Expert Systems International
1150 First Ave.
King of Prussia, PA 19406
(215) 337-2300

MPROLOG P500
Logicware
5000 Birch St.
Newport Beach, CA 9260
(714) 476-3634

LPA Micro-PROLOG Professional
Programming Logic Systems
31 Crescent Dr.
Milford, CT 06460
(203) 877-7988

PROLOG-86
Solution Systems
335B Washington St.
Norwell, MA 02061
(617) 659-1571

(continued from page 37)

mean this soft standard when I refer to PROLOG.) Turbo Prolog is a declarative language built around a resolution-based inference engine. The implementation uses recursion and backtracking and supports the usual PROLOG flow-of-control tools of *cut* and *fail* and the explicit ordering of clauses and conjuncts to control program flow. Turbo Prolog also allows the programmer additional control via a compiler directive that checks for possible nondeterministic clauses. It supports, with some exceptions and extensions, Clocksin and Mellish syntax.

Perhaps the greatest shortcoming of Turbo Prolog as an implementation of PROLOG is the lack of what might be called "metalinguistic" functions and operators. Some of these supplied in PROLOG are *arg, functor, name, op, clause*, and *call* and the *univ* operator. In general, these functions allow the PROLOG program to examine itself, to operate on code as data, and to construct new clauses and goals undreamt of by the programmer.

Of these, some are less important than others; *op* allows extension of the operators of the language so that, for example, you could define ^ to be the exponentiation operator that Borland didn't supply. This function is a handy tool, allowing the programmer to define the arity and structure (for example, two-argument infix) for new operators. But as Clocksin and Mellish point out, this capability is ultimately nothing more than a device for prettying up I/O; it adds no additional computational power.

Also, to some extent the metalinguistic predicates and operators are interchangeable or can be defined in terms of one another. You don't absolutely need the *univ* operator if you have both *functor* and *arg*, and vice versa. But either the *univ* operator or the combination of *functor* and *arg* is needed to map data structures and clauses (code elements) into one another so that in PROLOG code is truly data.

As an example of the power of these metalinguistic constructs, consider the following metainterpreter for PROLOG, adapted from Henryk Jan Komorowski and Shigeo Omori's article "A Model and an Implementation of a Logic Programming Environment," in the *Proceedings of the ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments*. This code is also adapted from Clocksin and Mellish.

```
prove(true) :- !.
prove(P, <morePs>) :-
        prove(P), prove(<morePs>).
prove(P) :-
        clause(P, Body), prove(Body).
```

These clauses mean succeed when the argument is true; to prove a conjunction, prove the first clause, then prove the rest; and to prove one thing, find a clause in the database with that thing as its head and prove the body of the clause. This metainterpreter allows the programmer to redefine the action of the PROLOG interpreter, and it's the function clause, with its ability to examine code as data, that permits this.

This predicate *prove* is a simple version of the PROLOG *call*. The Turbo Prolog manual (p. 151) also defines an interpreter (called *call*) that uses Turbo Prolog's call-by-reference capability. The listing (unfortunately complicated by several typographical errors) suggests how you might build the missing metalinguistic elements.

Nevertheless, the claim that Turbo Prolog is not full PROLOG is justified, and that places limits on what you can do with it. Complex applications developed under existing PROLOG versions may require significant rethinking before they can be ported to Turbo. On the other hand, Turbo Prolog does provide something that other PROLOGs may not in its support of DOS calls and machine-language functions. This low-level support suggests that anything missing from the compiler can be supplied—if you're willing to write it yourself.

What may prove to be as great a hindrance to experienced PROLOG programmers using Turbo Prolog, though, are the features the language has that, from a purist's point of view, it shouldn't have. These features are also the greatest advantage Turbo Prolog might claim over more faithful implementations.

Features in "core" PROLOG and not in Turbo Prolog	Features in Turbo Prolog and not in "core" PROLOG
"metalinguistic" features: =.. (univ), arg, call, clause, functor, gensym, name, op. I/O: get0, get, put, read, tab, display, tell, telling, told, user, see, seeing, seen, reconsult.	arithmetic: bitand, bitnot, bitxor, bitor, bitleft, bitright. I/O: readln, readchar, readint, readreal, readterm, writedevice, writef, about a dozen file functions, and a wealth of screen handling functions. string handling and type conversion: about a dozen functions. system access: bios, system, membyte, memword, portbyte, ptr_dword, storage, date, time.

Table 1: *Major feature differences between "core" PROLOG and Turbo Prolog*

```
goal
makewindow(1,7,7,"Source",0,0,20,35),
        write("Which file to copy ?"),
        cursor(3,8),readln(X),
makewindow(2,7,7,"Destination",0,40,20,35),
        write("What name for the copy ?"),
        cursor(3,8),readln(Y),
        concat(X," ",X1),concat(X1,Y,Z),
        concat("copy ",Z,W),
makewindow(3,7,7,"Process",14,15,8,50),
        write(" Copying ",X," to ",Y), cursor(2,3),
        system(W).
```

Table 2: *A Borland routine that gives users a window to DOS*

Claim 2: New Language

Turbo Prolog is not PROLOG at all but

Now available worldwide:

Prospero's professional language compilers for PCs and STs

PRO PASCAL & PRO FORTRAN-77:

FOR IBM PCs, XTs, ATs & COMPATIBLES, & ALL OTHER MS-DOS MACHINES. ALSO FOR ATARI ST, SINCLAIR QL

USE THE LANGUAGES THE PROFESSIONALS USE: PASCAL AND FORTRAN.
C is high on performance but low on safety, structure, portability and maintainability. Pascal is excellent for education and for long-term projects. Fortran gives you access to hundreds of existing programs – and uses established programming skills!

ISO-PASCAL
Pro Pascal is validated to ISO 7185 / ANSI 770X3.97 Class A (no errors) on the Z80 processor under CP/M and the 8088 processor under PC-DOS. This provides a guarantee that the compiler is complete and works perfectly.

FORTRAN-77
Pro Fortran-77 is a full implementation of ANSI Fortran–77 with no omissions and many useful extensions. Validation is imminent.

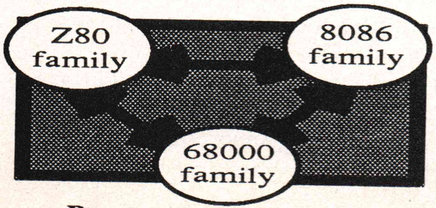
Prospero Compilers Work!
No known bugs when we ship - so you don't have to program around the holes.

"Lazy I/O" for interactive use.

Good Housekeeping
All files closed on exit from procedure

Hand-coded library
Produces compact and efficient programs.

Ideal for
- software developers
- universities and colleges
- government and industry
- students of computing
- development of personal skills
- solving technical problems
- training institutions



Prospero compilers: portable source code

16 digit accuracy
Single and double precision IEEE format arithmetic gives 7 or 16 digit precision.

50,000 Lines Big
Compiles big programs >50,000 lines... >5,000 identifiers...Separate compilation to build libraries and massive program suites.

Jumbo
New Jumbo memory model with Pro Fortran-77 gives unrestricted access to all 640K under MS-DOS.

Compilers include
- compiler
- linker
- run-time libraries
- librarian
- X-ref program
- sample programs
- 200+ page manual

GEM
Full GEM AES and VDI bindings supplied with Atari products and (on request) with Pro Pascal for MS-DOS.

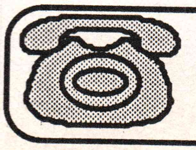
Amstrad & Z80
Pro Pascal and Pro Fortran-66 are available for Z80 machines.

Symbolic debugger
16 bit versions include Probe - a superb symbolic debugger. You can view fully compiled code as source, backtrack, display and change variables using source names, check the calling stack, and break anywhere. Pro Fortran-77 version can also Debug and produce execution profiles.

Pascal «—» Fortran
Interlinkable code means you can use the best of both languages. Ask for details.

We Specialize
At Prospero we specialize in writing language compilers. We produce the best possible compilers, conforming to the appropriate standard, and giving programmers a secure base on which to build.

Quote: "Pro Pascal is not only ISO-validated, it is also a superb-quality and very full software development tool." Personal Computer World Nov 1985



*Call early!
Before 1pm East coast,
10am West coast

Prospero compilers are used at more than 6000 sites around the world.

It's easy to order!
Credit card holders phone 011-441-741-8531. Mastercard, Visa, Diners & Amex accepted. Prospero's compilers are also available from good software dealers and distributors worldwide.

Reviews & Info
Call 011-441-741 8531 or write in for a free info pack with datasheets and magazine reviews.

New: Pro Fortran-77 for MS-DOS will be available in August/September 86.

Software distributors
Contact us for our Worldwide Distribution Guide.

US Distributors:

PC:	**PC Brand 212-410-4001**
	Software Consulting 215-837-8484
Atari:	**Apex Resources 617-232-9686**
	Oreman Sales 504-468-2001
QL:	**A+ Computer Resources 603-357-1800**
	Quantum 201-328-8846

Call for worldwide distributor list

Prospero Software
LANGUAGES FOR MICROCOMPUTER PROFESSIONALS
190 CASTELNAU, LONDON SW13 9DH, ENGLAND TEL 01-741 8531 TELEX 8814396

Mail order service
Send this coupon with check to Prospero Software Ltd, 190 Castelnau, London SW13 9DH, England. Add $5 shipping and allow 28 days for delivery.
Please supply (enter number required):
____ Pro Fortran-77 for Atari ST at $149
____ Pro Pascal for Atari ST at $149
____ Pro Fortran-77 for Sinclair QL at $129
____ Pro Pascal for Sinclair QL at $129
____ Pro Fortran-77 for MS-DOS at $495
____ Pro Pascal for MS-DOS at $390
____ Pro Pascal for CP/M-86 at $390
Special offer till 10/1/86: both Atari compilers for $269
I enclose payment of $____
Name..
Address.......................................
....
State/ Zip....................................
Telephone No..................................
In case of query call 011-441-741 8531 or telex 8814396. Tick product name if you just require information! DD 9/86

Circle **no. 160** on reader service card.

(continued from page 38)

a new language.

"It's not PROLOG; it's Turbo Prolog."—apocryphal Borland programmer.

The right-hand column of Table 1 points out another way in which Borland's compiler deviates from PROLOG: by extending it and by being in some senses a richer language. Turbo Prolog allows access to memory, I/O ports, and BIOS routines via a *bios* predicate and to DOS via a predicate called *system*. Using the *system* predicate and the strong display facilities of Turbo Prolog, the programmer can easily give the user a window to *DOS*. The program shown in Table 2, page 38, is a Borland-supplied, window-oriented, file copy routine that shows the *system* predicate in action. As you see, it is virtually devoid of any PROLOG declarative flavor.

Because Turbo Prolog is compiled, certain features Clocksin and Mellish describe, such as *trace*, are implemented in Turbo as compiler directives. Turbo Prolog also has some compiler directives expressly conceived to help assess efficiency of program structure in PROLOG, such as *check_determ*, *check_cmpio*, *nowarnings*, and *diagnostics*. These tools, which do such things as eliminating tail recursions and flagging possibly nondeterministic clauses, are needed in a language that provides as few explicit controls on program flow as does PROLOG. Turbo Prolog also maintains the programmer's variable names for postcompilation editing of source code.

You can, according to Borland (I haven't tested this), incorporate in your Turbo Prolog programs subroutines written in 8088 assembly language, FORTRAN, C, or Pascal (but not Turbo Pascal, yet). That alone won't make the routines PROLOG-like, of course. If you want to write components in some other language and have them perform PROLOG purposes, you may have to work a little: a single three-argument predicate could conceivably require nine separate routines if implemented in assembly language because each permutation of instantiated and uninstantiated arguments could require different action.

Some of Turbo Prolog's deviations from PROLOG style—for example, Turbo Prolog's strict data typing—force rethinking of program logic. You could argue that strict data typing violates PROLOG design and limits usefulness and portability. Borland's manual defends the practice in terms of creating a more secure program development environment and reducing space requirements for the language.

There are other arguments for using some kind of data typing in PROLOG: Daniel Brand's article "On Typing in Prolog," in the January 1986 *ACM SIGPLAN Notices* presents arguments for at least one model of typed PROLOG. Brand claims his model has the advantages of improved program readability, reduced computation time because of fewer and shorter clauses and reduced search space, and reduced need for the *cut* operator. It requires an enhanced unification algorithm that checks data types before unifying clauses. This slows things down a little, but Brand thinks the cumulative effect is faster code. Turbo Prolog's approach is different but shares some of these advantages.

A related limitation of Turbo Prolog is the constraint that you can only assert facts, not rules, and this also restricts the product's usefulness.

Is Turbo Prolog truly PROLOG? If you need the full metalinguistic power of the core predicates that Clocksin and Mellish sketch out in Chapter 6 of their book, or if you will be porting a complex existing PROLOG system to the PC—no. Otherwise, consider this just a particularly deviant dialect among other deviations. The reason for Turbo Prolog's deviations from Clocksin and Mellish is clear: the programmers wanted to make it fast.

Claim 3: Faster

Turbo Prolog is faster than the Japanese fifth-generation language systems.

This claim comes from the Turbo Prolog manual (p. 4): "Turbo Prolog runs on a computer costing about

"When am I ever going to see a fast C compiler with a full library–including ANSI library support–great documentation, plenty of support libraries and UNIX System V™ compatibility?"

Bill Davidsen, Software Engineer
Office Automation Group
General Electric Research and Development

"It's Microsoft C, and I can get it to you the day after tomorrow."

Bruce Lynch, President
The Programmer's Shop

Informed buying shouldn't take longer and cost more. Bill doesn't buy his software on the spur of the moment. Nor does he make a purchase without evaluating a ton of packages. But he refuses to make a full-time job of it.

So he lets us do a lot of the groundwork for him.

The Programmer's Shop offers Bill over 62 programming language implementations and 174 support programs, all from the same source. But we also give him informed product evaluations and recommendations based on his particular needs. And we offer a quarterly insider's newsletter on programming and industry trends, new tools and techniques.

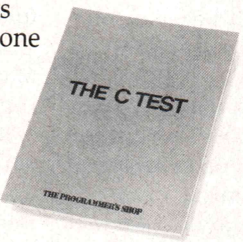
It's nothing that we don't do for all of our 10,000 customers.

We recommend evaluating software by getting detailed information from several different sources, including unbiased reports and reviews.

Is Microsoft C the best on the market? For Bill, sure. He wanted its tight code and needed its UNIX System V compatibility. And now Microsoft C Version 4.00 includes Microsoft CodeView,™ a source-level windowing debugger. Bill also liked the idea that we carry over 37 compatible support products for it. Including support libraries like CTree,™ Halo,™ Panel™ and PforCe.™ In the process, Bill learned quite a bit about C compilers.

C for yourself. Bill helped us compile a 16-page report of the objective opinions of 4 magazines, 14 users and 3 industry analysts: *The C Test.*

As an unbiased evaluation by users and professionals alike, *The C Test* is one of the most comprehensive and informative reports currently available on C development tools. *It's only available from The Programmer's Shop. And it's yours free for the asking.* Here's what you'll find in it:



The C Test ▪ Detailed Tech Specs
▪ Benchmark Source Code ▪ Magazine Reviews
▪ Users' Feedback ▪ Performance Benchmarks
▪ User Study and Profiles ▪ Test Drive Survey Results
▪ 37 Compatible Products

The C Test will give you the satisfaction of *knowing* you made the right choice from among all of the alternatives.

Call us for more than a bargain. Call us for advice. The Programmer's Shop specializes in making life easier, more informed and less expensive for people like Bill Davidsen. And you. We've become a success by giving the best advice for free and selling the best software for less.

Call us now with any question about your next software purchase. Given your needs and budget, we'll tell you what we suggest. Free.

To order Microsoft C ($319) or for your free copy of *The C Test,* simply call the toll-free number below:

1-800-421-8006. In Massachusetts, call 1-800-442-8070.

MICROSOFT. C Compiler Version 4.00

MICROSOFT C COMPILER
- Produces fast executables and optimized code including elimination of common sub-expressions. NEW!
- Implements register variables.
- Small, Medium and Large Memory model libraries.
- Compact and HUGE memory model libraries. NEW!
- Can mix models with NEAR, FAR and the new HUGE pointers.
- Library routines implement most of UNIX™ System V C library.
- Start-up source code to help create ROMable code. NEW!
- Full proposed ANSI C library support (except clock). NEW!
- Link your C routines with Microsoft FORTRAN (version 3.3 or higher), Microsoft Pascal (version 3.3 or higher) or Microsoft Macro Assembler.
- Microsoft Windows support and MS-DOS 3.1 networking support.

MICROSOFT PROGRAM MAINTENANCE UTILITY. NEW!
- Rebuilds your applications after your source files have changed.
- Supports macro definitions and inference rules.

OTHER UTILITIES.
- Library Manager.
- Overlay Linker.
- EXE File Compression Utility.
- EXE File Header Utility.

MICROSOFT CodeView
WINDOW-ORIENTED SOURCE-LEVEL DEBUGGER. NEW!
- Watch the values of your local and global variables and expressions as you debug.
- Set conditional breakpoints on variables, expressions or memory; trace and single step.
- Watch CPU registers and flags as you execute.
- Debug using your original source code, the resulting disassembly or both intermingled.

Microsoft C comes with a 30-day money-back guarantee from The Programmer's Shop.

THE PROGRAMMER'S SHOP

The programmer's complete source for software, services and answers.

128 Rockland Street, Hanover, MA 02339 (617)826-7531

Circle **no. 175** on reader service card.

(continued from page 40)

$2000, yet, in a comparison made in 1984 using an earlier version of the system, it produced programs that executed faster than those produced by the prototype of the Japanese Fifth Generation computer." And from the March 3 press release, Turbo Prolog "outperforms other existing PROLOG language tools by factors of up to 10,000."

The developers of the compiler that Borland is marketing in the United States as Turbo Prolog sacrificed power and portability for speed. How much speed did they buy?

The speed of AI languages is measured in *LIPS* (logical instructions per second). It's reasonable that a fifth-generation language should have a different unit for measuring performance from that used in evaluating third-generation languages. The first-through fifth-generation language classification is chiefly a matter of the power of a typical instruction. A fifth-generation logical instruction ought to do more than a third-generation instruction does—so the theory goes. Turbo Prolog has many instructions that PROLOG lacks, but these are all third-generation (or possibly, in the case of the graphics operations, fourth-generation) instructions.

To give a true picture of what *LIPS* measure as compared with non-logical *IPS* would require fairly detailed analysis of some large tasks, lifting out components that are and are not good candidates for backtracking and resolution. The picture would be complicated by subtle differences in the goals of the declarative and imperative modes of programming: How do you compare speed if you don't agree about what constitutes acceptable user input, for example?

What I'm doing here is much less than this; I'm providing the results of a typical benchmark for imperative programming with a rough bound on the number of logical instructions it requires in Turbo Prolog.

The results don't support Borland's extravagant claims. The benchmark I ran was a simple recursive version of the sieve of Eratosthenes. Generating all primes less than n should take on the order of n^2 operations or less: for primes less than 100, that's on the order of 10,000 LIPS as a rough bound. The Turbo Prolog result, 0.5 second for all primes less than 100, should be compared with results for your favorite imperative language and for other PROLOG implementations. (I got a value of 19 seconds for another, particularly slow, interpreted implementation of PROLOG.) Turbo Prolog was fast, but nowhere near 10,000 times as fast as the slowest competitor I could find. Furthermore, because the Symbolics PROLOG machine is projected to run at 100,000 LIPS, I don't think that any claim of speed for Turbo Prolog comparable to speed for serious fifth-generation machines need be taken seriously.

Finally, Turbo Prolog lacks virtual memory, a feature of some other microcomputer implementations. I suspect that the fast benchmark results Borland alludes to are based on accesses to databases in RAM.

Turbo Prolog requires an IBM PC or compatible with PC-DOS or MS-DOS 2.0 or later and 384K RAM. I tested it with an AT&T 6300 with 640K RAM. I also brought it up on a MacCharlie Plus with 640K RAM as a 128K Switcher application on a 512K Hyperdrive Mac.

Borland's speed claims are perhaps extravagant, but Turbo Prolog certainly produces fast code, particularly when databases are small enough to reside in RAM.

Claim 4: Difficult

PROLOG is difficult to understand, learn, and use, and Turbo Prolog is in this respect an implementation of PROLOG.

This one is common among experienced programmers. I support Morein's law, passed by Robert Morein, the author of A.D.A. PROLOG, a year or so ago, which states the following: If it's hard in FORTRAN, it's easy in PROLOG. If it's hard in PROLOG, it's easy in FORTRAN.

I think, but will not try to defend this opinion, that PROLOG is no harder than FORTRAN. I find Turbo Prolog to be an excellent learning environment, although it's arguable just what the learner is learning when learning Turbo Prolog.

Turbo Prolog has an excellent user interface, with multiple windows for editing, tracing, output, and messages and pull-down menus for accessing DOS, configuring the system, and selecting the destination for compilation (RAM, disk). It's a compiler with the interactive feel of an interpreter. It has the same editor Borland puts in all its products—fine if you like WordStar, but more to the point, instantly familiar to millions of people. This is the user interface Borland will put into the next version of Turbo Pascal, projected for release in the second quarter of next year.

The manual is a decent tool for learning Turbo Prolog or the beginnings of PROLOG. To go beyond, a book on the language is necessary. The manual contains abundant examples of code, both short illustrative segments and full programs. After developing the difficult topic of the *cut* operator, the manual gives practical, rule-of-thumb advice for its use. The manual's chief defects as a learning tool are an inadequate index, excessive errors, and the lack of adequate acknowledgment of deviations from Clocksin and Mellish core PROLOG. For that matter, I could find no references to Clocksin and Mellish or any other book on PROLOG. Nevertheless, it's generally good tutorial documentation and, if cleaned up for the next printing, should be useful to most people interested in getting started with PROLOG.

I conducted an informal test of ease of learning, teaching a novice the rudiments of Pascal and of PROLOG using the Turbo Pascal and Turbo Prolog manuals as texts. Although I have taught Pascal professionally and know it much better than I do PROLOG, I saw no difference in ease of acquiring the concepts.

Claim 5: Sorry Excuse for Code

Turbo Prolog is a "sorry excuse for coding" from "a couple of Danish folks."

PC Week made this claim in the May 27, 1986, issue under its house pseudonym Spencer F. Katt.

It's true that Turbo Prolog is a Danish product, as is Turbo Pascal; nearly all Borland products are European. Borland is oddly secretive about the fact that it is primarily in the business of software acquisition and distribution; Borland's Independent Contractor Nondisclosure Agreement (which I have not signed) defines as trade secret "all information concerning the identity or whereabouts of key devel-

(continued from page 42)

opers of Company products, past or current." Reflex may be the one Borland product widely known to have been acquired.

It's not true, however, that Turbo Prolog is poorly written. This is a good piece of coding. It produces surprisingly fast compiled code. The user interface is better than Turbo Pascal's current interface for rapid development of small routines. And the links to DOS and other languages and the tools for optimization make this product more than a toy.

The applications for which Turbo Prolog is most appropriate may be applications that are on the border of artificial intelligence rather than in the mainstream. As the manual suggests, it could serve as a good specification language. The speed of the compiled code and the fifth-generation richness of the instructions may make it a good database development language, whereas the lack of metalinguistic power may make it a poor candidate for abstract problem solving. In any case, it has the Turbo Pascal strengths for quick development and testing of small modules. The product has a place.

Claim 6: Half-Million Sales

Borland will sell half a million copies of Turbo Prolog in the next two years.

That's what Philippe Kahn claims. Well, actually he says there may be that many people using the product—not the same thing, quite.

It's possible. Logic programming is in vogue, and Borland has built a reputation with Turbo Pascal that could transfer to Turbo Prolog. Novice Turbo Pascal users didn't care about its deviations from the standard, and novice Turbo Prolog users will be even less sensitive to the less-constraining soft standard for PROLOG. Of course, Turbo Prolog is not the only PROLOG, nor this time has Borland got the cheapest product. But if any PROLOG is as successful as Kahn hopes his will be, it will be good news for all PROLOG developers.

Some companies are beginning to look upon Borland as the company that opens markets for them. On May 20, less than three weeks after Turbo Prolog became available, Arity announced its trade-in plan. Buy Arity's $795 PROLOG development system, and the firm will give you $50 for page 213 of your Turbo Prolog manual as proof of purchase (that's the page that explains that there is no simple way of interfacing Turbo Prolog modules with Turbo Pascal programs). The program is due to expire about the time you read this, but it presents one view, the view Arity would like you to have, of Turbo Prolog's position: a beginner's language for those who want to play with PROLOG syntax before deciding whether to move up to a serious development system. It's the view Logitech would like you to have of Turbo Pascal vis-à-vis Logitech's Modula-2, which is why it is selling a product that lets you translate your old Turbo Pascal programs to Modula-2.

There is undoubtedly some truth to this perception. It's quite possible that Turbo Prolog, with its inviting user interface, will open up a large market for PROLOG products.

Logic programming has been projected to be the dominant form of programming in the next century. The next century begins in 14 years and 4 months. That should be more than enough time to turn PROLOG into that "important new generation of programming languages and systems" Clocksin and Mellish envision, especially if Kahn's prediction of half a million new PROLOG programmers is even half right.

Bibliography

Byte (August 1985). Special section on declarative languages.

Clark, K. L., and McCabe, F. G. *micro-PROLOG: Programming in Logic.* Englewood Cliffs, N.J.: Prentice-Hall, 1984.

Clocksin, W. F., and Mellish, C. S. *Programming in Prolog.* Berlin: Springer-Verlag, 1984.

Cohen, Paul R., and Feigenbaum, Edward A. *The Handbook of Artificial Intelligence*, vol. 3. Stanford, Calif.: HeurisTech Press, 1982.

Communications of the ACM (September 1985). Special section on architectures for knowledge-based systems.

de Saram, Hugh. *Programming in micro-PROLOG.* Chichester, England: Ellis Horwood Ltd., 1985.

Dr. Dobb's Journal (March 1985). Special *PROLOG* issue.

Ennals, Richard. *Beginning micro-PROLOG.* New York: Harper & Row, 1984.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

High-Speed Thrills

A Review of Eight Turbo Boards for the IBM PC

by Mike Elkins and Steve King

We used six applications programs and five benchmark programs to test the boards.

Over the past few years, most of us switched from our old Z80 computers using the CP/M-80 operating system to IBM PCs (or clones) using PC(MS)-DOS. Then we experienced the big disappointment of learning that the new machines weren't quite as fast as the old ones were. After all, most of us, even programmers, have the great American dream: "More speed, faster is better, drag you for pink slips at the next stoplight!" Finding that your new computer isn't as fast as your old one can damage your ego more than learning your brand new, expensive, foreign sports car isn't as fast as your old jalopy.

When IBM announced the PC/AT, we envisioned the enhanced productivity and throughput the original PC had seemed to promise but had not produced. After all, we can rationalize that the most time-consuming part of program development is the endless debug-recompile-test process. If we have faster machines, we can shorten the software development process and we can save money. The AT rekindled the American dream: lightning fast compiles; the pleasure of being power users. Most of us, though, have invested too much time and money expanding our old PCs and PC/XTs with memory expansion boards, hard disks, and anything else we could stuff into them to really justify junking our old PCs and XTs to buy new ATs.

Early 1985 produced rumors of high-speed computer chips and boards that could speed up PCs to match the AT's performance—well, almost match the AT's performance. NEC V20 rumors promised to replace your old 8088, provide complete software compatibility, and give you an 85 percent increase in processing speed—still no match for the AT but not bad. In reality, the V20 provided what may be an 18 percent speed increase rather than the dreamed of 85 percent gain.

Last fall, we wandered through Comdex in Las Vegas and saw prototype accelerator boards that made PCs run not just as fast as ATs but even faster—the boards were software compatible with both machines. These boards are now becoming one of the most popular and desired PC and XT hardware enhancements. In this article, we'll review six of these so-called turbo boards and maybe rekindle your American dream.

The Benchmarks

Most turbo boards utilize on-board high-speed memory with a 16-bit interface, a high clock rate, and a more efficient CPU to produce the desired increase in throughput. The boards we tested for this article produced processing speeds ranging from one and a half times the speed of a standard PC to well beyond the speed of a standard AT—even beyond the speed of an AT with a higher clock speed.

Our test machine was an IBM PC containing:

- 256K RAM on the motherboard
- AST Six Pac with 384K RAM
- two floppies
- 10-megabyte internal drive
- 10-megabyte IOMEGA Bernoulli Box
- STB Graphics Plus and Color Monitor
- Mitsuba Expansion Unit

We used the following programs to test the turbo boards' PC compatibility: BASICA, Brief Programmer's Editor, dBASE III, Framework II, Microsoft C 3.0, and Microsoft Link. We've noted where any didn't function properly.

We used the following benchmark programs compiled with the Microsoft C 3.0 compiler:

- Compile/Link: Compile and link a 425-line C program with a RAM disk for the TMP work area.
- Sieve * 10: Ten loops through the Eratosthenes Sieve prime-number program (Listing One, page 88).
- Sieve * 100: 100 loops through the Sieve program.
- Dhrystone: The Dhrystone benchmark program (Listing Two, page 88).
- The Sysinfo utility included in Norton's Utilities, Version 3.1.

The public-domain utility TIMEIT, written by Jack Means, timed the first

Mike Elkins and Steve King, 720 Flora Dr., Oceanside, CA 92056

three benchmarks. Table 1, below, contains a summary of our results.

Reinhold P. Weicker originally developed the Dhrystone program using the Ada programming language. A careful C conversion preserves the Ada likeness at the expense of C format and at the same time demonstrates the versatility of C. The program does nothing useful but is a well-rounded benchmark program containing dynamic memory allocation and manipulation, along with some number crunching.

The Boards

We'll now give a brief description of each of the boards we used in our benchmark tests and some brief comments on their performance. See the list of companies and addresses on page 49.

QuadSprint

Quadram's QuadSprint contains an 8086 microprocessor running at 9.54 MHz. This board takes up one full-size slot and replaces the 8088 with a jumper-type ribbon cable. Because you remove the 8088 from the PC, you have to de-install the board in order to return to native mode. Quad Sprint needs no software drivers or special programs for operation.

The QuadSprint board has 4K RAM, high-speed cache memory that is software switchable with a small BASIC program, which is listed in the manual. The program uses an *OUT* command to send a value to the specified port (on or off). The board was software and hardware compatible with all our tests. QuadSprint uses the existing PC memory via the 8-bit PC system bus, which is probably why the speed increase was not as high as that of some of the other 8086 boards.

PC Turbocharger

PC Turbocharger from Univation uses an 8086 microprocessor running at 10 MHz. The board takes up a full slot and replaces the 8088 with a ribbon cable. The company provides an 8087 noise suppressor to plug into the 8087 socket on the motherboard. You must replace two chips on the board if you have the PC Model 2 with a 256K motherboard. Failure to read the documentation prior to installation to ensure the proper chips are in place may scramble the data on your hard disk!

The PC Turbocharger board doesn't require that you reboot the system to change speeds, as is the case with some of the other boards.

Univation provides several utilities with the PC Turbocharger, including a RAM disk, spooler, and cache—all have user-definable sizes. The company also includes a memory tester and a program that copies ROM to high-speed RAM for fast execution of programs that make BIOS calls, such as BASICA. Because this board contains its own memory, we neutralized the AST Six Pac memory by setting its memory jumpers to 0.

SpeedPac 286

SpeedPac 286 from Victor Technology contains an 80286 microprocessor running at 7.2 MHz. The board takes up one half-size slot and replaces the 8088 with a jumper-type ribbon cable. You can't switch back to standard mode, and you must set jumpers to indicate available memory. Because the board has no memory, only PC memory is used. The board does, however, contain a high-speed 8K cache buffer and resident caching software. No software drivers or special programs are necessary. The SpeedPac 286 has a socket for an 80287 math coprocessor.

Victor Technology provides special instructions for installing dBASE III. Because you can't switch back to PC mode, many types of protected software may require that you remove the board temporarily when installing the software on your hard disk. This is a minor inconvenience in exchange for the added performance you receive for this board's low price. The board is compatible with IBM's

	Comp/Link index	Comp/Link seconds	Sieve*10 index	Sieve*10 seconds	Sieve*100 index	Sieve*100 seconds	Dhrystone index	Dhrystone loops/sec	Sysinfo index
IBM PC	1.00	231.95	1.00	12.30	1.00	107.60	1.00	333	1
IBM PC/AT		N/A		N/A		N/A	3.13	1041	5.60
PC Turbo 286	3.35	69.15	3.44	3.57	4.55	23.62	5	1666	8.40
PC Turbocharger	1.83	110.48	1.90	6.45	2.39	44.99	2.63	877	2.20
Pfaster 286	3.30	70.19	3.79	3.24	5.33	20.16	5.17	1724	8.40
QuadSprint	1.51	153.41	1.62	7.58	1.92	56.08	1.97	657	2
SpeedPac 286	2.08	111.45	2.37	5.17	3.43	31.36	3.41	1136	6.60
286 Speed Pack		N/A	2.28	5.39	3.28	32.80	3.84	1282	7

Table 1: *Benchmark summary*

HIGH SPEED THRILLS
(continued from page 47)

Enhanced Graphics Adapter.

We found that the 8088 replacement cable was too short to reach over an existing board in the first slot. We had to insert the SpeedPac 286 board in the slot closest to the 8088 socket, which is not a disadvantage in the XT where the half slot is seldom used. This board performed very well considering that it was using existing 200-ns PC memory and an 8-bit interface. Victor Technology also offers a 60-day money-back guarantee. This is a real plus!

286 Speed Pack

Classic Technology Corp.'s 286 Speed Pack contains an 80286 microprocessor; its speed is not documented. The board takes up one full-size slot and replaces the 8088 with a jumper-type ribbon cable. You plug the PC's 8088 into a socket on the board and change back to standard mode with a switch on the back of the board. Then you must reboot. The socket is positioned so that the 8088's label reads in the opposite way from the labels for the rest of the chips on the board; this could cause confusion, even though it is well documented.

You must set a jumper on the 286 Speed Pack board to indicate motherboard type: Model 1, 2, or XT. The board has a socket for an 80287 math coprocessor and has its own RAM, which may be expanded to 4 megabytes. This board is designed primarily for the XT and, if you have an internal hard disk, requires a 130-watt power supply. Because our test computer had only a 60-watt power supply and an internal hard disk, we couldn't run the PC with the hard disk installed. We also didn't run the compile/link benchmark. When installed in an XT, 286 Speed Pack uses the first 64K RAM on the motherboard to hold DOS; all other applications run in the high-speed memory located on the board itself.

The 286 Speed Pack board's benchmark times were inconsistent, as was the case with most of the boards; we used the "best case" values. Classic Technology Corp. also provides network boards that allow multiple workstations to be attached to a PC/XT containing 286 Speed Pack. This setup gives you an XT file server with the power and speed of an AT server.

Pfaster 286

Pfaster 286 from Phoenix Computer Products Corp. uses an 80286 microprocessor running at 8 MHz. The board takes up a full slot and is the easiest of all the boards to install. It uses no ribbon cables, and you leave the 8088 in place. Software and easily installed device drivers activate it. You switch to the standard PC mode with a program called PSLOW and back to the 80286 mode with PFAST. Our test board came with 2-megabyte RAM, but it is also available with only 1 megabyte. The board also contains an 80287 math coprocessor.

Pfaster 286 supports the Lotus/Intel/Microsoft (LIM) Extended Memory Specification (EMS) and comes with a RAM disk that utilizes the extra megabyte of memory for non-EMS use.

The Pfaster 286 board was the fastest in many of the tests, but the screen I/O seemed jumpy and especially slow with programs such as Framework II that write directly to the video memory. The Norton Sysinfo results were inconsistent, ranging from 6.1 to 8.4 in the PFAST mode.

In the PFAST mode, Pfaster 286 uses motherboard memory for caching, allowing the compile/link benchmark to outperform an 8-MHz IBM PC/AT containing a high-speed hard disk. The compile time was much better (51.74 seconds) when a 1-megabyte RAM disk was used to hold all the necessary files. CHKDSK revealed that 704K RAM was available after the 1-megabyte RAM disk was defined and the 8088 memory was set aside for caching.

We couldn't convince the Brief editor to run in the high-speed mode—the cursor disappeared and characters were lost or duplicated. We recommend this board, which seems well implemented and versatile despite the I/O problem, but it's expensive.

PC Turbo 286

Orchid Technology's PC Turbo 286 uses an 80286 microprocessor run-

ning at 8 MHz with no wait states. The board takes up a full slot but requires no extra cables or chip removal. You activate this board with software device drivers that are easy to install and customize. The installation software carefully changes your existing AUTOEXEC.BAT file and doesn't disturb any commands, paths, or prompt commands. We found the software installation procedures to be very professional; they prompted for parameters and always explained the available options. The test board came with 1-megabyte RAM, but it is available with an additional 1 megabyte on a daughterboard. It fully supports the LIM EMS.

PC Turbo 286's benchmark times were very consistent, and all test software performed quite well. Screen I/O was extremely fast, and there were no noticeable differences when we used the Turbo mode. An installation option allows an increased screen I/O speed for nonflicker graphics boards—for example, our test PC's STB board. With the software installed for the flicker option, we found the resulting screen I/O was a little slower and somewhat jumpy, but PC Turbo 286 performed much better than did Pfaster 286.

Fixed-disk caching (floppies, too, with a command-line option) takes advantage of the memory on the motherboard while the board is operating in the Turbo mode. RAM disks and spoolers in the standard PC memory are also accessible in the Turbo mode.

PC Turbo 286 supports the 80287 math coprocessor, but our evaluation unit didn't have one. The board performed flawlessly with all hardware in the test machine.

Orchid Technology's unit can also run in the PC/AT as a true dual processor. Two PC Turbo boards in a PC or XT can provide the same effect. The software to provide access to the other boards was in development and not available (but was documented) for our tests, however.

Orchid Technology's documentation is complete and helpful, which is surprising because we had a beta version of the board. The documentation covers jumper settings to change I/O addresses and interrupt request lines to minimize chances of unresolved hardware conflicts.

Try It, You'll Like It

Using these boards, we found microprocessor speed increases from two to more than eight times the speed of the 8088 in a stock PC but at best about a five times' increase in actual throughput. Standard I/O devices provide the primary bottleneck. Because of software enhancements, such as cache and RAM-disk programs that come with most of the boards, you can achieve the actual throughput of an IBM PC/AT and greatly surpass the standard PC. We see these turbo boards as serious alternatives to an AT because they provide the desired performance increase and save money—it's much cheaper to upgrade your old PC or XT with one of these boards than to buy an AT, even one of the new, cheap AT clones.

We tested two categories of boards—Intel 8086-based (or NEC V30 compatible) and Intel 80286-based. The first group benchmarked very closely, and choosing a winner was difficult.

All the installed 80286 boards performed consistently faster than a stock IBM PC/AT. For raw performance and overall throughput and compatibility, Orchid Technology's PC Turbo 286 was a clear winner, far surpassing the standard IBM PC/AT's performance. It has lightning-fast screen I/O and a reasonable price for an AT alternative, and it fully supports EMS. If price is the primary consideration, look at Victor Technology's SpeedPac 286. It can attain the speed of the AT without expensive memory replacement—its perfor-
With public-domain cache and RAM-disk software, this board could provide the throughput you're looking for.

DDJ

(Listings begin on page 88.)

Vote for your favorite feature/article.
Circle Reader Service **No. 5.**

PC Turbo 286
Orchid Technology
47790 Westinghouse Dr.
Fremont, CA 94539
(415) 490-8586
Price: $1,195 (1 megabyte)
$1,480 (2 megabytes)
Reader Service Number 45

PC Turbocharger
Univation Inc.
1037 N. Fair Oaks Ave.
Sunnyvale, CA 94089
(408) 745-0180
Price: $595 (128K)
$795 (640K)
Reader Service Number 46

Pfaster 286
Phoenix Computer Products Corp.
320 Norwood Park South
Norwood, MA 02062
(617) 762-5030
Price: $1,495 (1 megabyte)
$1,895 (2-megabyte unit evaluated)
$2,395 (2 megabytes w/ViaNet LAN software)
Reader Service Number 47

QuadSprint
Quadram
4355 International Blvd.
Norcross, GA 30093
(404) 923-6666
Price: $645
Reader Service Number 48

SpeedPac 286
Victor Technology
980 El Pueblo Rd.
Scotts Valley, CA 95066
(408) 438-6680
Price: $595
Reader Service Number 49

286 Speed Pack
Classic Technology Corp.
2090 Concourse Dr.
San Jose, CA 95131
(408) 434-9333
Price: $995
Reader Service Number 50

Listing One (continued from July)

```
      FDB HERE-6          Link to HERE
NOT3  LDX SP
      LDA SP0,X
      COMA
      STA SP0,X
      INCX
      LDA SP0,X
      COMA
      STA SP0,X
      JMP NEXT
*
      FCB 2               1+
      FCC '1+ '
      FD , NOT3-6         Link to NOT
ONEP  LDX SP
      INCX                point to low byte
      LDA SP0,X
      ADD #1
      STA SP0,X
      LDX SP              now the high byte
      LDA SP0,X
      ADC #0
      STA SP0,X
      JMP NEXT
*
      FCB 3               HLD
      FCC 'HLD'
      FDB ONEP-6          link to 1+
HLD3  LDA #HLD            (fall through to DOUSE)
*
DOUSE ADD #USER               Does the common part of the
      LDX SP                  execution of a user variable
      DECX
      STA SP0,X
      CLRA
      DECX
      STA SP0,X
      STX SP
      JMP NEXT
*
      FCB 5               STATE
      FCC 'STA'
      FDB HLD3-6          link to HLD
STA5  LDA #STATE
      BRA DOUSE
*
      FCB 7               CONTEXT
      FCC 'CON'
      FDB STA5-6          link to STATE
CON7  LDA #CONTEXT
      BRA DOUSE
*
      FCB 7               CURRENT
      FCC 'CUR'
      FDB CON7-6          link to CONTEXT
CUR7  LDA #CURRENT
      BRA DOUSE
*
      FCB 5               FORTH
      FCC 'FOR'
      FDB CUR7-6          link to CURRENT
FOR5  LDA #FORTH
      BRA DOUSE
*
      FCB 1               !
      FCC '! '
      FDB FOR5-6          link to FORTH
STO   LDX SP              move addr to Load
      LDA SP0,X
      INCX
      STA LOAD+1
      LDA SP0,X
      INCX
      STA LOAD+2
      LDA SP0,X           now move data to addr
      INCX                high byte first
      STX SP
      CLRX
      JSR LOAD
      LDX SP
      LDA SP0,X           now the low byte
      INCX
      STX SP
      LDX #1
      JSR LOAD
      JMP NEXT
*
      FCB 2               C!
      FCC 'C! '
      FDB STO-6           link to !
CSTO  LDX SP              move addr to Load
      LDA SP0,X
      INCX
      STA LOAD+1
      LDA SP0,X
      INCX
      STA LOAD+2
      INCX                drop high data byte
      LDA SP0,X           and move low byte
      INCX
      STX SP
      CLRX
      JSR LOAD
      JMP NEXT
*
      FCB 1
      FCC ',  '
      FDB CSTO-6          link to C!
COMA  LDA DP              move DP to Load
      STA LOAD+1
      LDA DP+1
      STA LOAD+2
      LDX SP              move data to DP
      LDA SP0,X           high byte
      INCX
      STX SP
      CLRX
      JSR LOAD
      LDX SP              low byte
      LDA SP0,X
      INCX
      STX SP
      LDX #1
      JSR LOAD
      LDA #2
INCDP ADD DP+1            bump DP
      STA DP+1
      LDA DP
      ADC #0
      STA DP
      JMP NEXT
*
      FCB 2               C,
      FCC 'C, '
      FDB COMA-6
CCOMA LDA DP
      STA LOAD+1
      LDA DP+1
      STA LOAD+2
      LDX SP              move data to DP
      INCX                drop high byte
      LDA SP0,X           get low byte
      INCX
      STX SP
      CLRX
      JSR LOAD
      LDA #1              bump DP by 1
      BRA INCDP
*
      FCB 3               DUP
      FCC 'DUP'
      FDB CCOMA-6         link to C,
DUP3  LDX SP
      LDA SP0,X           get high byte
      DECX                and bump SP to point to new location
      DECX
      STA SP0,X           then store it
      LDX SP
      INCX                get low byte
      LDA SP0,X
      DECX                bump SP for it too
      DECX
      STA SP0,X           and store it
      DECX                update SP
      STX SP
      JMP NEXT
*
      FCB 2               +!
      FCC '+! '
      FDB DUP3-6          link to DUP
PLSTO LDX SP              move Addr to Load and Get
      LDA SP0,X
      INCX
      STA LOAD+1
      STA GET+1
      LDA SP0,X
      INCX
      STA LOAD+2
      STA GET+2
      STX SP
      LDX #1              get low byte of addr data
      JSR GET
      LDX SP              get low byte of number
      INCX
      ADD SP0,X
      LDX #1              and save it back
      JSR LOAD
```

(continued on page 52)

Listing One (listing continued)

```
      CLRX                      The same for the high byte
      JSR GET
      LDX SP
      ADC SP0,X
      CLRX
      JSR LOAD
      INC SP                    update SP
      INC SP
      JMP NEXT
*
      FCB 6                     LATEST
      FCC 'LAT'
      FDB PLSTO-6               link to +!
LAT6  JMP DOCOL
      FDB CUR7
      FDB FTCH
      FDB FTCH
      FDB EXIT
*
      FCB 5                     ALLOT
      FCC 'ALL'
      FDB LAT6-6                link to LATEST
ALL5  JMP DOCOL
      FDB DP2
      FDB PLSTO
      FDB EXIT
*
      FCB 3                     LIT
      FCC 'LIT'
      FDB ALL5-6
LIT3  LDA IP                    move IP to Get
      STA GET+1
      LDA IP+1
      STA GET+2
      LDX #1                    move low byte to stack
      JSR GET
      LDX SP
      DECX
      STA SP0,X
      STX SP
      CLRX                      and then the high byte
      JSR GET
      LDX SP
      DECX
      STA SP0,X
      STX SP
      LDA #2                    now bump IP
      ADD IP+1
      STA IP+1
      CLRA
      ADC IP
      STA IP
      JMP NEXT
*
QIMM  LDX SP                    Tests for IMMEDIATE
      INCX                      using count byte
      LDA SP0,X                 from <FIND>
      TSTA
      BMI QID
      CLRA
      BRA QSKIP
QID   LDA #$FF
QSKIP STA SP0,X
      DECX
      STA SP0,X
      JMP NEXT
*
*
MESS  FCB 67
      FCC 'RAFOS '
      FCC 'FORTH '
      FCC 'V1.0'
*
      FCB CR
      FCB LF
      FCC 'A TEAM ROSSBY PRODUCTION'
*
      FCB CR
      FCB LF
      FCC '(C) EVERETT CARTER 1986'
*
LOK   FCB 3
OK    FCC ' OK'                 The FORTH prompt
*
*
*     DEFAULT OUTER INTERPRETER
*
*
OUTER FDB COU5
      FDB TYPE
      FDB INLINE
      FDB DFND
      FDB ZBRAN
      FDB $001E
```

```
      FDB QIMM
      FDB NOT3
      FDB ZBRAN
      FDB $0010
      FDB STA5
      FDB FTCH
      FDB ZBRAN
      FDB $0008
      FDB COMA
      FDB BRAN
      FDB $FFE6
      FDB EXE7
      FDB BRAN
      FDB $FFE0
      FDB HERE
      FDB NUM8
      FDB ZBRAN
      FDB $0014
      FDB STA5
      FDB FTCH
      FDB ZBRAN
      FDB $FFD0
      FDB COMP
      FDB LIT3
      FDB COMA
      FDB BRAN
      FDB $FFC6
      FDB QUES
      FDB BRAN
      FDB $FFBA
*
*          POWER ON RESET ROUTINE
*
      FCB 4                     COLD
      FCC 'COL'
      FDB LIT3-6                link to LIT
*
COLD  BSET3 $05
      BSET3 PUT
      LDX #$3F                  Move the default RAM data
SDAT  LDA ROM,X
      STA 0,X
      DECX
      CPX #$20
      BNE SDAT
      LDX #$80                  Move the self-modifying code
SREPT LDA ROM,X                 to its executable location
      STA 0,X                   (done in two steps to avoid the
      INCX                       CPUs stack: 40-7F)
      BNE SREPT
      CLRX
SREP2 LDA ROM+$100,X            (moving 200 HEX bytes)
      STA $100,X
      DECX
      BNE SREP2
*
*     Calculate the HIGH and LOW BYTES of OUTER
*
HO    EQU OUTER/$100*$100
LO    EQU OUTER-HO
HO1   EQU OUTER/$100
*
      LDA #LO                   Load the default
      STA START+1               Outer Interpreter
      LDA #HO1                  into START
      STA START
*
*     Calculate the HIGH and LOW BYTES of Latest entry
*
HCR   EQU LATEST/$100*$100
LCR   EQU LATEST-HCR
H1CR  EQU LATEST/$100
*
      LDA #H1CR                 Initialize FORTH
      STA USER+FORTH
      LDA #LCR
      STA USER+FORTH+1
*
*
*     Calculate the HIGH and LOW BYTES of MESS
*
H     EQU MESS/$100*$100
L     EQU MESS-H
H1    EQU MESS/$100
*
      CLRX
      LDA #L                    Push start up message
      DECX
      STA SP0,X
      LDA #H1
      DECX
      STA SP0,X
      STX SP                    Initialize Stack Pointer
```

(continued on page 54)

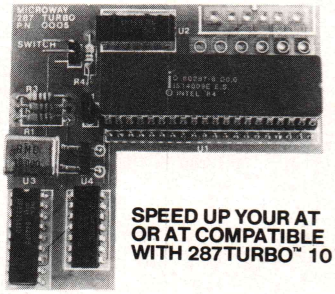
A MEGABYTE FOR DOS!

MicroWay is the world's leading retailer of 8087s and high performance PC upgrades. We stock a complete selection of 8087s that run from 5 to 12 MHz. All of our coprocessors are shipped with a diagnostic disk and the best warranty in the business – one year! We also offer daughterboards for socketless computers (NEC PC) and 287Turbo which increases the clock speed of the 80287 from 4 to 10 MHz. Our NUMBER SMASHER/ECM™ runs at 12 MHz with a megabyte of RAM and achieves a throughput of .1 megaflops with 87BASIC/INLINE, Intel Fortran, or Microsoft Fortran. Software reviewers consistently cite MicroWay software and 8087 expertise as the best in the industry! Our customers frequently write to thank us for recommending the correct software and hardware to meet their specific needs. They also thank us for our same-day shipping! In addition to our own products which support the 8087 and 80287, we stock the largest supply of specialized software available. For more information call us at

617-746-7341

NUMBER SMASHER/ECM™
THE FASTEST ACCELERATOR CARD AVAILABLE

gives you 12 MHz speed in two modes: 704K or one megabyte of "Extended Conventional Memory." MEGASWITCH MMU and MegaDOS software make it possible to run DOS applications with up to 1015K using PC compilers, AutoCAD and Lotus 1-2-3. Does not require EMS software. Totally compatible. Priced from **$599** with 512K to **$1199** for complete package. Optional 8087-12 ... **$295**



SPEED UP YOUR AT OR AT COMPATIBLE WITH 287TURBO™ 10 MHz

Micro Way®
8087 Support

For the IBM PC, PC XT, PC AT and Compatibles.

A2D-160™ MicroWay's Data Acquisition Board performs 160,000 12 bit Analog to Digital conversions per second! Includes software drivers. The fastest 12 bit A to D board available. For the IBM PC XT and compatibles. **$1295**

87SFL™ MicroWay's Scientific Function Library contains 170 scientific and engineering functions. Callable from most 8087 compatible compilers ... First Language **$250**; Additional **$100**.

MATRIXPAK™ manages a **MEGABYTE!** Written in assembly language, our runtime package accurately manipulates large matrices at very fast speeds. Includes matrix inversion and the solution of simultaneous linear equations. Callable from RM or MS Fortran, MS Assembler, or 87BASIC/INLINE. .... each **$99**

87FFT™ Written in assembly language, performs Forward and Inverse FFTs on real and complex arrays which occupy up to 512 Kbytes of RAM. Also does convolutions, auto correlations, hamming, complex vector multiplication, and complex to radial conversions. Callable from most 8087 compatible compilers. **$200**

87FFT-2™ performs two-dimensional FFTs. Ideal for image processing. Requires 87FFT **$100**

FASTBREAK™ employs the 8087 to increase the speed of Lotus 1-2-3™ Version 1A or 1A* by up to 36:1 .... **$79**

87Verify™ For users who have to be absolutely sure of their results! This background task periodically performs an 8087 accuracy and stress test. **$49**

Microsoft Fortran V 3.31	$209
IBM Professional Fortran	$565
Ryan-McFarland Fortran V 2.0	$399
NAG Fortran Library	$300
Grafmatic for Fortran or Pascal	$125
MultiHalo Graphics (1 language)	$189
LABTECH NOTEBOOK	$745
UnkelScope	$549
INTEL ABOVE BOARD	CALL
JRAM, AST, MAYNARD	CALL

MegaPage™ The only Intel-Lotus EMS board which comes with two megabytes of cool-running, low power drain CMOS RAM installed. Includes RAM disk, print spooler, disk cache, and EMS drivers. For the IBM PC, XT & compatibles. ... **$549**

MegaPage AT/ECC™ EMS card for the PC AT and compatibles includes Error Correction Circuitry. With ECC, 11 RAM chips cover 256K so the user never encounters RAM errors. With 1 megabyte CMOS **$799**; with 3 megabytes CMOS **$1295**. Optional serial/parallel daughterboard. .... **$95**.

DFixer™ Our disk utility which thoroughly checks PC or AT hard disks for bad sectors and updates the MS DOS file allocation table accordingly. Solves the AT hard disk problem! ... **$149**

DOptimizer™ Optimizes the way your hard disk or floppy stores its files. Speeds up accesses by recombining fragmented files. **$49**

DCache™ Our disk caching software speeds up your I/O by storing repetitively used tracks in memory. The amount of memory used can be selected in 64 Kbyte banks **$49**

87MACRO/DEBUG™ Contains all the pieces needed for writing 8087/80287 assembly code & MicroWay's 87DEBUG debugger. **$199**

OBJ→ASM™ A multipass object module translator and disassembler. Produces assembly language listings which include public symbols, external symbols and labels commented with cross references. Ideal for patching object modules for which source is not available. **$200**

87BASIC™ includes patches to the IBM BASIC or MS Quick BASIC Compiler for USER TRANSPARENT 8087 support. Provides super fast performance for all numeric operations including trigonometrics, transcendentals, addition, subtraction, multiplication, and division. ..... each **$150**

87BASIC/INLINE™ converts the output of the IBM BASIC Compiler into optimized 8087 inline code which executes up to seven times faster than 87BASIC. Supports separately compiled inline subroutines which are located in their own segments and can contain up to 64 Kbytes of code. This allows programs greater than 128K! Requires the IBM BASIC Compiler Version 1 and a Macro Assembler. Includes 87BASIC ... **$200**

MICROWAY UDI runs RTOS or RMX compilers under DOS **$300**

8087 UPGRADES

All MicroWay 8087s include a one year warranty, complete MicroWay Test Program and accurate installation instructions.

8087 5 MHz **$109**
For the IBM PC, XT and compatibles.

8087-2 8 MHz **$149**
For Wang, AT&T, DeskPro, NEC, Leading Edge.

80287-3 5 MHz **$179**
For the IBM PC AT and 286 compatibles.

80287-6 6 MHz **$229**
For 8 MHz AT compatibles.

80287-8 8 MHz **$295**
For the 8 MHz 80286 accelerator cards.

NEC V20, V30	**$16, $30**
64K RAM Set 150ns	**$10**
256K RAM Set 150ns	**$29**
256K RAM Set 120ns	**$39**
128K RAM Set PC AT	**$49**

287Turbo™ 10 MHz If you own an AT, Deskpro 286 or AT compatible, this is the card you need to get reasonable numeric performance. It plugs into your 80287 socket and includes a specially driven 10 MHz 80287. The card comes in three configurations. The IBM AT version includes a hardware RESET button **$450**

287Turbo 8 MHz **$369**

87/88Turbo™ is a stubby card which includes a clock calendar and a speed controller which changes the speed of your motherboard from 4.77 to 7.4 MHz. Its use requires your PC to have a socketed 8284. Typical speed increase is 1.6 to 2.0. The card overcomes slow hardware by slowing up only when such devices are accessed and running at full speed otherwise **$149**
Optional 8087-2 .... **$149**

286TurboCache™ This new MicroWay accelerator uses 8K of cache memory and 80286/80287 processors to provide an average speed increase of 3:1 for most programs. Call for specifications and benchmarks **$595**

Call for our complete catalog of software which supports the 8087.

In London, please phone 223-7762

Micro Way

P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

Listing One (listing continued)

```
.WARM  LDA #$80            Initialize input terminators
       STA TIB+$7E
       STA TIB+$7F
       CLR USER+STATE       Put system in EXECUTION state
       CLR USER+STATE+1
       CLR RP               Initialize Return Stack pointer
       JSR CRLF
       LDA START            Load the IP
       STA IP
       LDA START+1
       STA IP+1
       JMP NEXT             GO...
*
*
       FCB 4                  SWAP
       FCC 'SWA'
       FDB COLD-6             link to COLD
SWAP   LDX SP
       LDA SP0,X
       INCX
       STA PH
       LDA SP0,X
       INCX
       STA PL
       LDA SP0,X
       INCX
       STA QH
       LDA SP0,X
       STA QL
       LDA PL
       STA SP0,X
       LDA PH
       DECX
       STA SP0,X
       LDA QL
       DECX
       STA SP0,X
       LDA QH
       DECX
       STA SP0,X
       JMP NEXT
*
       FCB 3                  SP!
       FCC 'SP!'
       FDB SWAP-6             link to SWAP
SPSTO  CLR SP
       JMP NEXT
*
*
*
*      S E R I A L  I/O  R O U T I N E S
*
*
*
*      GETCHAR/GETC --- GET A CHARACTER FROM THE TERMINAL
*
*      A GETS THE CHARACTER TYPED, X IS UNCHANGED
*
GETC      STX XTEMP
GETCHAR   EQU GETC
          LDA #8
          STA COUNT
GETC4     CLI
          SEI
          BRSET2 PUT,GETC4
          LDA PUT
          AND #!11
          TAX
          LDX DELAYS,X          load Baud delay
GETC3     LDA #4
GETC2     DECA
          BNE GETC2
          TSTA
          DECX
          BNE GETC3
          BRSET2 PUT,GETC4
          TST ,X
          TST ,X
GETC7     BSR  DELAY
          BRCLR2 PUT,GETC6
GETC6     TST ,X
          ROR CHAR
          DEC COUNT
          BNE GETC7
          CLI
          BSR DELAY
          LDA CHAR
          AND    #$7F           Mask the eighth bit.
          LDX XTEMP
          RTS
*
*
*
*      OUTCHAR/PUTC --- PRINT A ON THE TERMINAL
*
*      X AND A UNCHANGED
*
PUTC      STA CHAR
OUTCHAR   EQU PUTC
          STA ATEMP
          STX XTEMP
          LDA #9
          STA COUNT
          CLRX
          CLC
          SEI
          BRA PUTC2
PUTC5     ROR CHAR
PUTC2     BCC PUTC3
          BSET3 PUT
          BRA PUTC4
PUTC3     BCLR3 PUT
          BRA PUTC4
PUTC4     JSR DELAY,X
          DEC COUNT
          BNE PUTC5
          BSET2 PUT
          BSET3 PUT
          CLI
          BSR DELAY
          LDX XTEMP
          LDA ATEMP
          RTS
*
*
*      WAIT --- PRECISE DELAY
*                A AND X ARE ZERO AT EXIT.
*
WAIT      LDA    #1             ADJUST FOR FIRST TIME
DELAY     EQU WAIT
          AND #!11
          TAX
          LDX DELAYS,X
          LDA #$F9
DEL3      ADD #$08
DEL2      DECA
          BNE DEL2
          TSTX
          BSET1 PUT
          DECX
          BNE DEL3
          LDA #0
          RTS
*
DELAYS    FCB  $20           300 BAUD
          FCB  $08          1200 BAUD
          FCB  $01          9600 BAUD
*
*
CRLF      LDA #CR
          JSR OUTCHAR
          LDA #LF
          JSR OUTCHAR
          RTS
*
       FCB 2                  CR
       FCC 'CR '
       FDB SPSTO-6            link to SP!
CR2    BSR CRLF
       JMP NEXT
*
       FCB 6                  CREATE
       FCC 'CRE'
       FDB CR2-6              link to CR
CRE6   JMP DOCOL
       FDB BL2
       FDB WORD
       FDB LIT3
       FDB #04
       FDB ALL5
       FDB LAT6
       FDB COMA
       FDB CUR7
       FDB FTCH
       FDB STO
       FDB EXIT
*
*
*
*
*****************************************************************
*
*      INTERRUPT VECTORS
*
        ORG    MEMSIZ-10      START OF VECTORS
*
        FDB    WTIME          TIMER IRQ VECTOR FROM WAIT STATE
        FDB    WTIME+3        ALTERNATE TIMER VECTOR
        FDB    WTIME+6        IRQ VECTOR.
        FDB    WARM           SWI TO FORTH INITIALIZATION POINT
        FDB    COLD           POWER ON VECTOR
*
        END
```

End Listing

Listing One (Text begins on page 10.)

```
Program Squareroot;
{
  Squareroot algorithm & testprogram; DDJ March 1986, p.122

  Features:  - sqrt routine in 68000 machine language;
             - long integer loopcount;
}

const       { Iteration count for test loop }
            NNR = 6E4; { real, for printing of statistics }
            NNS = '60000';        { string, for assignment to long integer }

type        long = record
                        low,high : integer;
                    end;

var         finished   : boolean; { flag for loop }
            number, limit    : long;              { loop count, loop limit }
            sqrrt,                                { square root result }
            sqrrto,                               { last displayed square root }
            t1,t2               : integer; { parameters for system time }
            time1, time2        : real;    { start, end time }

{ --- ROUTINES FOR LONG (32-BIT) INTEGER SUPPORT --- }

procedure lg_clr(var l:long);     external;
{ Clears long integer l }

procedure lg_asn_DU(s:string; var l:long);    external;
{ Assigns the unsigned decimal string s to the long integer l }

procedure lg_incl(var l:long);      external;
{ Increments long integer l by 1 }

procedure lg_grt(a,b:long; var flag:boolean);    external;
{ Compares long integers a and b and assign result to flag }

{ --- SQUAREROOT ROUTINE --- }

procedure sqrt(number:long; var result:integer);   external;
{ Calculates square root of 'number' and returns it in 'result' }

begin { main }

  sqrrto := 100;

  lg_clr(number);      { number := 0 }
  lg_asn_DU(NNS,limit); {limit := NNS }
  finished := false;

  write('Start...');
  time(t1,t2);  time1 := t2;   { get start time }

  while not finished do { calculate sqrt(number) }
    begin
      sqrt(number,sqrrt);
    {
      if sqrrt <> sqrrto
        then begin
            write ('Number = ',number.high:6,' | ',number.low:6);
            writeln('  ---  Sqrt = ',sqrrt:4);
            sqrrto := sqrrt;
        end;
    }
      lg_incl(number);                        { number := number + 1 }
      lg_grt(number,limit,finished);      { finished := (number > limit) }
    end;

  time(t1,t2);    time2 := t2;   { get end time }

  writeln('finished !'); writeln;
  writeln('Time: ',(time2-time1)/60,' seconds')

end.
```

End Listing One

Listing Two

```
*
* Squareroot algorithm;  DDJ March 1986, p.122
*
* 68000 assembly language version
*
* Features: - equivalent to compiler-generated code;
*
*
* procedure sqrt(number:long; var result:integer);
*
* Calculates integer square root of 'number' and returns it in 'result';
*
*
* Register usage:
* --------------
*
* D0 : word register                    A0 : parameter stack pointer
* D1 : number                   A1 : scratch register
* D2 : guess1
* D3 : guess2
* D4 : error
*
        proc      sqrt,2                      ;2 words of parameters
*
* Get parameters from stack
*
        move.l    (sp)+,a0   ;get return address
        move.w    2(sp),a1   ;get ^number
        move.l    (a1),d1                    ;get number
*
* bra Q15 ;--- for timing only
*
        beq       Q8                          ;if number=0, skip
*
* Set initial values
*
Q7      moveq     #1,d2                       ;guess1 := 1
                  move.l     d1,d3                      ;guess2 := number
*
* Do shifts until guess1 ~  guess2
*
Q9      move.l    d2,d0                       ;guess1 to work register
        asl.l     #1,d0                       ;guess1 * 2
        cmp.l     d3,d0                       ;compare with guess2
        bge       Q11                         ;branch if guess1 * 2 >= guess2
        asl.l     #1,d2                       ;guess1 := guess1 * 2
        asr.l     #1,d3                       ;guess2 := guess2 / 2
        bra       Q9
*
* Now do divisions
*
Q11
Q13     add.l     d3,d2                       ;guess := guess1 + guess2
        asr.l     #1,d2                       ;guess1 := (guess1+guess2)/2
        move.l    d1,d0                       ;number to work register
        divs      d2,d0                       ;number / guess1
        move.w    d0,d3                       ;guess2 := number/guess1
        ext.l     d3                          ;extend to 32 bits
        move.l    d2,d0                       ;guess1 to work register
        sub.l     d3,d0                       ;guess1-guess2
        move.l    d0,d4                       ;error := guess1-guess2
        ble       Q14                         ;if error <= 0
        bra       Q13                         ;loop back if error > 0
Q14     move.l    d2,d0                       ;result := guess1
        bra       Q15
Q8      moveq     #0,d0                       ;result := 0
*
* Set result & return to caller
*
Q15     movea.w (sp)+,a1      ;get ^result
        move.w    d0,(a1)                     ;store result
*
        addq.l    #2,sp                       ;drop ^number
        jmp       (a0)                        ;return to caller
*
        .nolist
```

End Listing Two

Listing Three

```
*
* Squareroot algorithm; DDJ March 1986, p.122
*
* 68000 assembly language version
*
* Features:  - hand-optimized machine code;
*
*
* procedure sqrt(number:integer; var result:integer);
*
* Calculates square root of 'number' and returns it in 'result';
*
*
* Register usage:
* --------------
*
* D0 : work register, error               A0 : ^result
* D1 : number                             A1 : ^number
* D2 : guess1,result
* D3 : guess2
* D4 : temporary storage for return address
*
*
        .proc      sqrt,2                     ;2 words of parameters
*
* Get parameters from stack
*
        moveq #0,d2                           ;result := 0 --- remove for timing
*
        move.l     (sp)+,d4   ;get return address
        move.w     (sp)+,a0   ;get ^result
        move.w (sp)+,a1                       ;get ^number
        move.l     (a1),d1                    ;get number
*
* bra Q15 ;--- for timing only
*
        beq        Q15                        ;if number=0, then result=0
*
* Set initial values
*
Q7      moveq      #1,d2                      ;guess1 := 1
                   move.l      d1,d3                   ;guess2 := number
*
* Do shifts until guess1 ~  guess2
*
Q9      asl.l      #1,d2                      ;guess1 := guess1 * 2
        cmp.l      d3,d2                      ;compare with guess2
        bge        Q11                        ;branch if guess1 * 2 >= guess2
        asr.l      #1,d3                      ;guess2 := guess2 / 2
        bra        Q9
*
Q11     asr.l      #1,d2                      ;adjust guess1
*
* Now do divisions
*
Q13     add.l      d3,d2                      ;guess1 := guess1 + guess2
        asr.l      #1,d2                      ;guess1 := (guess1+guess2)/2
        move.l     d1,d3                      ;guess2 := number
        divs       d2,d3                      ;guess2 := number / guess1
        ext.l      d3                         ;extend guess2 to 32 bits
        move.l     d2,d0                      ;guess1 to work register
        sub.l      d3,d0                      ;error := guess1 - guess2
        bgt        Q13                        ;loop back if error > 0
*
* Store result and return to caller
*
Q15     move.w     d2,(a0)                    ;store result
*
        movea.l    d4,a0                      ;move return address to adr-reg
        jmp        (a0)                       ;return to caller
*
        .nolist
```

End Listing Three

(continued on page 58)

Western Computer 286 Turbo™

Standard Features

- IBM PC/AT Compatible with 512K RAM
- Up to 1 MB of system memory on the main board
- Switch selectable 8 or 10 MHz operation with one wait state on memory access (80286-10)
- One parallel port/one serial port and Clock/Calendar on main board
- Optional Serial Port on board
- Mass storage options from 20 to 140 MB Hard Disk Drives and 20 to 60 MB tape backup systems
- EGA video options available for CAD/CAM, word processing, etc. . . .
- One year factory warranty

Western Computer has high quality computers at very competitive quantity pricing.

Western Computer
17781 MITCHELL STREET, IRVINE, CA 92714 U.S.A.
PHONE (714) 553-1611
Customer Service Only (714) 553-1705
TELEX 7566731 - Answer Back WESTERN COMP
FAX (714) 553-0236

Western Computer XT Turbo™

Standard Features

- IBM PC/XT compatible with 640K RAM on main board
- 4.77 or 8 MHz operation (CPU board made in U.S.A.)
- Two 360K Floppy Disk Drives
- Hercules compatible monochrome graphics controller (720 x 350) or IBM compatible color graphics adapter (320 x 200 - four colors or 640 x 200 - two colors) and parallel printer port
- Monochrome monitor (Composite or TTL input)
- IBM PC/AT style keyboard
- Various hard disk drive and tape backup options available
- One Year Factory Warranty

Western Computer
Australiasia Limited
4-82 ABBOT STREET
ASCOT BRISBA
QUEENSLAND, AUSTRALIA
4007 - #(07) 268-6589
Telex: AA144746
FAX: #(07) 2685256
Answer Back MCGUIR

Western Computer

"For High Technology and Performance"

European Head Office
BELECTRONIC SA,
RUE CENTRALE 43
CH-1880-BEX, SWITZERLAND
PHONE (025) 631250
TELEX 456 168
Answer Back BELE CH.

Listing Four

(Listing continued, text begins on page 10.)

```
*
* Squareroot algorithm; DDJ November 1985, p.88
*
* 68000 assembly language
*
* procedure sqrt(number:long; var result:integer);
*
* Calculates the integer squareroot of 'number' and returns it in 'result'
*
*
* Register usage
* --------------
*
* D0 : number                  A0 : scratch, for pointers
* D1 : error term              A1 : scratch, for pointers
* D2 : estimate
* D3 : corrector term
* D4 : loop counter
*
*
        .proc     sqrt,2                   ;2 words of parameters
*
        move.w    6(sp),a0   ;get ^number
        move.l    (a0),d0                  ;get number into d0
*
* bra exit ;--- for timing only
*
        moveq     #16-1,d4   ;set loopcount,16*2 - 32 bits
        moveq     #0,d1                    ;clear error term
        moveq     #0,d2                    ;clear estimate
*
* Calculate squareroot
*
sqrtl asl.l #1,d0            ;get 2 leading bits,
        roxl.l    #1,d1                    ;one at a time, into
        asl.l     #1,d0                    ;the error term
        roxl.l    #1,d1
        asl.l     #1,d2                    ;estimate * 2
        move.l  d2,d3
        asl.l     #1,d3                    ;corrector - 2 * new estimate
        cmp.l     d3,d1
        bls       sqrt2                    ;branch if error term <- corrector
        addq.l    #1,d2                    ;otherwise, add low bit to estimate
        addq.l    #1,d3
        sub.l     d3,d1                    ;and calculate new error term
*
sqrt2 dbra        d4,sqrtl   ;do all 16 bit-pairs
*
* Set result & return to caller
*
exit    move.l    (sp)+,a0   ;get return address
        move.w    (sp)+,a1   ;get ^result
        move.w    d2,(a1)                  ;store integer result
        addq.l    #2,sp                    ;drop ^number
        jmp       (a0)                     ;return to caller
*
        .nolist
```

End Listing Four

Listing Five

```
djnz:                                             * 10    djnz e
        move.b      (pseudopc)+,d0                * d0 <-- distance
        subq.b      #1,regb(regs)                 * dec b
        beq         djnz2                         * loop count expired
        ext.w       d0                            * to word
        ext.l       d0                            * to long
        add.l       d0,pseudopc                   * add distance
djnz2:
        NEXT
jr:                                               * 18    jr e
        move.b      (pseudopc)+,d0                * d0 <-- distance
        ext.w       d0                            * to word
        ext.l       d0                            * to long
        add.l       d0,pseudopc                   * add distance
        NEXT
jrnz:                                             * 20    jr nz,e
        move.b      (pseudopc)+,d0                * d0 <-- distance
        btst        #6,regf                       * if Z bit set
        bne         jrnz2                         * then no branch
        ext.w       d0                            * to word
        ext.l       d0                            * to long
        add.l       d0,pseudopc                   * add distance
jrnz2:
        NEXT
jrz:                                              * 28    jr z,e
        move.b      (pseudopc)+,d0                * d0 <-- distance
        btst        #6,regf                       * if Z bit reset
        beq         jrz2                          * then no branch
        ext.w       d0                            * to word
        ext.l       d0                            * to long
        add.l       d0,pseudopc                   * add distance
jrz2:
        NEXT
jrnc:                                             * 30    jr nc,e
        move.b      (pseudopc)+,d0                * d0 <-- distance
        btst        #0,regf                       * if C bit set
        bne         jrnc2                         * then no branch
        ext.w       d0                            * to word
        ext.l       d0                            * to long
        add.l       d0,pseudopc                   * add distance
```

(continued on page 60)

NEW FEATURES IN LATEST RELEASE

VEDIT PLUS

MULTIPLE WINDOWS

POP-UP MENUS

KEYSTROKE MACROS

EXECUTE DOS PROGRAMS

For over six years VEDIT has been the choice of professionals who demand the most powerful editing software available. CompuView has once again enhanced this power with the latest VEDIT PLUS - you can now open windows to simultaneously edit several files, access editing functions with pop-up menus, use keystroke macros to speed editing and run other programs within VEDIT PLUS.

Whether your needs are program development, technical writing or word processing, VEDIT PLUS is your answer. VEDIT PLUS is simple enough to learn for the novice, yet has the speed, flexibility and power to satisfy the most demanding computer professional. Its powerful macro programming language helps you eliminate repetitive editing tasks.

If you take your editing seriously, you need VEDIT PLUS. With over 40,000 users, you can depend on VEDIT PLUS to perform consistently and reliably. As have GE, EDS, U.S. Navy, GM, Sperry and many others. Available for MS-DOS, PCDOS, CP/M-86 and CP/M-80. List price $185.

'To sum things up, VEDIT PLUS is a small, fast, sophisticated editor with a wealth of features and a good macro language. It offers many rewards for the dedicated programmer.'
Computer Language, Chris Wolf, Scott Lewis, Mark Gayman 6/86

'VEDIT PLUS is a wholly remarkable program: blindingly fast, extremely powerful, and highly flexible.'
Profiles Magazine, Robert Lavenda 4/86

VEDIT PLUS FEATURES

- Simultaneously edit up to 37 files of unlimited size.
- Split the screen into variable sized windows.
- 'Virtual' disk buffering simplifies editing of large files.
- Memory management supports up to 640K.
- Execute DOS commands or other programs.
- MS-DOS pathname and CP/M user number support.
- Horizontal scrolling - edit long lines.
- Flexible 'cut and paste' with 36 text registers.
- Customization - determine your own keyboard layout, create your own editing functions, support any screen size, any CRT.
- Optimized for IBM PC/XT/AT. Also 132 column & up to 70 lines.

EASY TO USE

- Interactive on-line help is user changeable and expandable.
- On-line integer calculator (also algebraic expressions).
- Single key search and global or selective replace.
- Pop-up menus for easy access to many editing functions.
- Keystroke macros speed editing, 'hot keys' for menu functions.

FOR PROGRAMMERS

- Automatic Indent/Undent for 'C', PL/I or PASCAL.
- Match/check nested parentheses, i.e. '{' and '}' for 'C'.
- Automatic conversion to upper case for assembly language labels, opcodes, operands with comments unchanged.
- Optional 8080 to 8086 source code translator.

FOR WRITERS

- Word Wrap and paragraph formatting at adjustable margins.
- Right margin justification.
- Support foreign, graphic and special characters.
- Convert WordStar and mainframe files.
- Print any portion of file; separate printer margins.

MACRO PROGRAMMING LANGUAGE

- 'If-then-else', looping, testing, branching, user prompts keyboard input, 17 bit algebraic expressions, variables.
- CRT emulation within windows, Forms entry.
- Simplifies complex text processing, formatting, conversions and translations.
- Complete TECO capability.
- Free macros: • Full screen file compare/merge • Sort mailing lists • Print Formatter • Main menu

CompuView

Circle **no. 122** on reader service card.

1955 Pauline Blvd., Ann Arbor, MI 48103 (313) 996-1299, TELEX 701821

Listing Five (Listing continued, text begins on page 10.)

```
jrnc2:
        NEXT
jrc:
        move.b   (pseudopc)+,d0        * 38 jr c,e
        btst     #0,regf               * d0 <-- distance
        beq      jrc2                  * if C bit reset
        ext.w    d0                    * then no branch
        ext.l    d0                    * to word
        add.l    d0,pseudopc           * to long
                                       * add distance
jrc2:
        NEXT
```

End Listing Five

Listing Six

```
/* c_draw.c */              /*   jnm  5-27-86  rev.1     */

/* line drawing routine using Bresenham's algorithm.  */


c_draw (x1, y1, x2, y2, color)
int x1, y1, x2, y2, color;

 {
int incl, inc2, inc3, xend, yend;
int d, x, y, dx, dy;

dx = abs (x2 - x1);
dy = abs (y2 - y1);
if (dy <= dx)
    {
    if (x1 > x2)
        {
        x = x2;
        y = y2;
        xend = x1;
        dy = y1 - y2;
        }
    else
        {
        x = x1;
        y = y1;
        xend = x2;
        dy = y2 - y1;
        }
    incl = dy << 1;
    inc2 = (dy - dx) << 1;
    inc3 = (dy + dx) << 1;
    d = (dy >= 0) ? incl - dx:incl + dx;
    while (x < xend)
        {
        pcvwd (y, x, color);        /* or whatever point plotting */
        x++;                        /* function you have handy... */
        if (d >= 0)
            {
            if (dy <= 0)
                d += incl;
            else
                {
                y++;
                d += inc2;
                }
            }
        else
            {
            if (dy >= 0)
                d += incl;
            else
                {
                y--;
                d += inc3;
                }
            }
        }
    }
else
    {
    if (y1 > y2)
        {
        y = y2;
        x = x2;
        yend = y1;
        dx = x1 - x2;
        }
    else
        {
        y = y1;
        x = x1;
        yend = y2
        dx = x2 - x1;
        }
    incl = dx << 1;
    inc2 = (dx - dy) << 1;
    inc3 = (dx + dy) << 1;
    d = (dx >= 0) ? incl - dy:incl + dy;
    while (y < yend)
        {
        pcvwd (y, x, color);
        y++;
        if (d >= 0)
            {
            if (dx <= 0)
                d += incl;
            else
                {
                x++;
                d += inc2;
                }
            }
        else
            {
            if (dx >= 0)
                d += incl;
            else
                {
                x--;
                d += inc3;
                }
            }
        }
    }
pcvwd (y, x, color);
}
```

End Listings

Circle **no. 227** on reader service card.

Listing One (Text begins on page 14.)

```
                              Listing 1 -- dtree.c
-------------------------------------------------------------------------------
#include <stdio.h>
#include <ctype.h>
#include <process.h>     /* needed by spawn()     */
#include <mydir.h>       /* needed by dir()       */
#include <signal.h>      /* needed by signal()    */

/* ------------------------------------------------------------------------
 * WHEREIS and DTREE
 *
 * A general-purpose directory traversal program. If invoked with name
 * "whereis" it searches for a file in the directory system. If invoked
 * with "dtree" it does the above and can also print the directory tree or
 * executes a program in each directory. See usage() and wusage() below
 * for more details about the command-line syntax.
 *
 *                      (C) 1986 Allen I. Holub.  All rights reserved.
 */

extern  DIRECTORY       *mk_dir( int );
extern  void            del_dir( DIRECTORY* );
extern  void            dir( char*, DIRECTORY* );
extern  char            *strrchr( char*, int );

/* ------------------------------------------------------------------------
 * These IBM graphics (box drawing) characters are used only if the
 * output stream is stdout and isatty() is true (it will be false if
 * stdout is redirected).
 *
 *       |                 |                 |
 * ELL   |       T_RIGHT   +----      VERT   |       (dash)  -----
 * \300 +----    \303      |          \263   |       \304
 */

static char     *Graph_chars[] = { "\263",  "\300\304\304\304\304\304",
                                            "\303\304\304\304\304\304"  };

static char     *Norm_chars[]  = { "|", "+-----", "+-----" };
static char     **Cset         = Norm_chars;

#define VERT    Cset[0]
#define ELL     Cset[1]
#define T_RIGHT Cset[2]

/* ----------------------------------------------------------------------- */

#define DSIZE  255
static  char   Startdir[DSIZE+1];    /* The cwd when the program started */
static  char   Map[ 64/8 ];          /* Bitmap for 64 bits. If the       */
                                     /* directory tree is deeper than    */
                                     /* this, we're in trouble.          */

static char     **Args       = NULL; /* Thing to execute, pass to spawnv */
static int      Short_pname = 0;     /* Use short pathnames              */
static int      Draw        = 0;     /* Draw directory tree              */
static char     *Findfile   = NULL;  /* File for which we're searching   */

/* ------------------------------------------------------------------------
 * bitmap routines:
 *              testbit(x)       Evaluates TRUE if bit x is set.
 *              setbut (x,val)   Set bit x if val is TRUE, else clear it.
 */

#define testbit(x)  ( Map[x >> 3] & (1 << (x & 0x07)) )

static  setbit( c, val )
int     c, val;
{
        if( val )
                Map[c >> 3]  |=   1 << (c & 0x07) ;
        else
                Map[c >> 3]  &=  ~(1 << (c & 0x07)) ;
}

/* ----------------------------------------------------------------------- */

pline( depth, terminate )
{
        /*      Print all the spaces and vertical bars in a graphic
         *      representation of a tree. Does nothing if Draw if FALSE.
         */

        int     i;

        if( !Draw )
                return;

        for(i = 0; i < depth-1 ; i++ )
                printf(testbit(i) ? "%s      " : "       ", VERT );

        if( terminate )
                printf("\n");
}

/* ----------------------------------------------------------------------- */

pname( dname )
char    *dname;
{
        /*      Print a directory name with or without the full path
         *      spec (depending on whether Short_pname is set.
         */

        char    *name;

        if( !Short_pname  || (dname[0] == '/' && !dname[1]) )
                name = dname;

        else if( name = strrchr(dname, '/') )
                name++;
        else
                name = dname;

        printf( "%s\n", name );
}

/* ----------------------------------------------------------------------- */

execute( dname )
char    *dname;
{
        /*      Execute the command specified on the command line from
         *      the directory we're now visiting. This routine changes
         *      the current directory but doesn't put it back.
         *      The Args vector must point at the command array.
         */

        if( Args )
        {
                chdir( dname );
                if( spawnvp(P_WAIT, *Args, Args) == -1 )
                        perror(*Args);
        }
}

/* ----------------------------------------------------------------------- */

find( dname )
char    *dname;
{
        /* Look for the Findfile file in dname. If it's there, print
         * the full path and file names and return 1, else return 0.
         */

        static char     pathname[DSIZE];
        char            **vects;
        int             count;
        int             rval;
        DIRECTORY       *dp;

        sprintf( pathname, "%s/%s", dname, Findfile );

        if( !(dp = mk_dir( 64 )) )
        {
                printf("dtree: Out of memory\n");
                return 0;
        }

        dp->files = 1;                  /* Get files only                 */
        dp->sort  = 1;                  /* sort the list                  */
        dp->path  = 1;                  /* and include the full path name */
        dir( pathname, dp );            /* Fill the DIRECTORY structure   */

        vects = (char **) dp->dirv;     /* ..., and print it.             */
        count =   rval =  dp->nfiles;
        while( --count >= 0 )
                printf( "%s\n", *vects++ );

        del_dir( dp );
        return rval;
}

/* ----------------------------------------------------------------------- */

static  prnt( dname, others )
char    *dname;
int     others;
{
        /* Does a recursive traversal of the directory tree rooted at
         * dname. "others" is true if the calling routine has more
         * subdirectories to print.
         */

        DIRECTORY       *dp;
        char            **vects;
        int             count;
        static int      depth = -1;

        if( ++depth && Draw )
        {
                pline( depth, 0 );
                printf("%s", others ? T_RIGHT : ELL );
        }

        if( Findfile )                          /* If -f was set don't    */
        {                                       /* execute a -e command   */
                if( find( dname ))              /* unless we've found the */
                        execute( dname );       /* file. find() prints the */
        }                                       /* filename if it exists. */
        else
        {
                pname   ( dname );              /* else print the directory */
                execute( dname );               /* name and execute the cmd */
        }

        if( !(dp = mk_dir( 32 )) )
        {
                printf("dtree: Out of memory\n");
                return;
        }

        dp->dirs = 1;                   /* Get subdirectories            */
        dp->sort = 1;                   /* and sort them.                */
        dp->exp  = 1;                   /* expand subdirectories rather than */
                                        /* printing their names.         */
```

(continued on page 66)

ONE STOP.

PROGRAMMER'S
QUALITY INTEGRITY
Service
SUPPORT PRICE
CONNECTION

We're Programmer's Connection, a leading independent dealer of quality programmer's development tools specifically for IBM Personal Computers. We're your best one-stop source for the professional PC / MS-DOS and XENIX programming tools you need.

Since we're an *independent* dealer, we'll look out for your best interest. Our courteous, knowledgeable, noncommissioned sales staff is always ready to assist you. If you aren't sure about your needs, you can talk to our experienced, professional technical staff for sound, unbiased advice. They can compare products, answer technical questions and send you detailed product information that's tailored to your needs.

Our product line consists of hundreds of high quality software development tools specifically for IBM Personal Computers and compatibles. We don't carry *everything* ever written for programmers — only those products that meet our very high standards for quality and value.

The products we carry are the latest version and they all come with the same manufacturer's technical support as if buying direct. Unlike other dealers who participate in the software gray market, we're authorized to sell *every* product we carry.

We discount all software products — even special order items. We don't try to fool you by discounting some products and charging full list price for others. Every product in our price list is shown with its discounted *and* retail price. We want you to know exactly how much you'll save on *every* product.

Other dealers add extra rush charges for shipping via express services. We'll express your order to you with no special handling charges. We only charge you what the shipping carrier charges. And if you have your order shipped via standard UPS, shipping is *FREE*.

Most popular products are in stock and ready for immediate shipment. Maintaining an adequate inventory is part of our philosophy of fast, efficient service. If we don't have a product in stock, we'll get it for you fast. Again — no extra charge.

Some dealers charge your credit card at the time they take your order. This means you could be left waiting in vain for your order for weeks or months while they use your money interest free. We *never* charge your credit card until we actually ship your order.

Quite simply, the discount prices you see on the next two pages are all you pay. There are *no* hidden charges. We don't charge extra for standard UPS shipping, credit cards, COD orders, purchase orders or special handling (except orders outside the U.S. and Canada are charged $5 for customs form preparation).

Our goal is customer satisfaction and that's why we offer 30 day no-risk return guarantees and 30 day evaluation periods on most of our products. Note that some items, especially those with source code, are restricted by the manufacturers from this guarantee. Please call for specific details.

Our customers keep coming back because we consistently provide high quality service and low discount prices. So make the connection today and see how convenient one-stop service can be.

CALL TOLL FREE

US: 800-336-1166 ▪ CANADA: 800-225-1166

OHIO AND OVERSEAS: 216-877-3781
Hours: 8:30 AM to 8:00 PM Eastern Time, Mon.-Fri.
Customer Service: 216-877-1110

programmer's connection

Call or write for our *FREE* comprehensive price guide. Turn the page for our latest advertised price list.

Circle **no. 129** on reader service card.

PROGRAMMER DEVELOPMENT TOOLS FOR THE IBM-PC/XT/AT
and compatibles.

apl language

	LIST	OURS
APL*PLUS/PC System by STSC	595	**449**
APL*PLUS/PC Tools Vol 1 by STSC	295	**239**
APL*PLUS/PC Tools Vol 2 by STSC	150	**129**
APL*PLUS Spreadsheet Manager by STSC	195	**159**
APL*PLUS/UNX System *For AT Xenix* by STSC	995	**795**
Btrieve *ISAM File Mgr with No Royalties* by SoftCraft	250	**195**
Financial/Statistical Library by STSC	275	**219**
Pocket APL by STSC	95	**79**
STATGRAPHICS *Statistical Graphics System* by STSC	795	**619**

arity products

	LIST	OURS
Arity Expert System Development Package ... *New*	295	**279**
Arity File Interchange Toolkit ... *New*	50	**48**
Arity Prolog Compiler & Interpreter ... *New*	795	**739**
Arity Prolog Interpreter ... *New*	350	**329**
Arity SQL Development Package ... *New*	295	**279**
Arity Screen Design Toolkit ... *New*	50	**48**
Arity Standard Prolog ... *New*	95	**89**

artificial intelligence

	LIST	OURS
ExpertEDGE by Human Edge	795	**659**
Experteach II *Complete System* by Intelligenceware	475	**CALL**
EXSYS *Expert System Development Software* by EXSYS	395	**339**
First Class by Human Edge ... *New*	500	**419**
GCLISP *Golden Common LISP* by Gold Hill	495	**CALL**
GCLISP 286 Developer *LM Interpreter & LM Compiler*	1190	**CALL**
Insight 1 *AI Primer* by Level Five Research	95	**75**
Insight 2+ by Level Five Research	485	**389**
Logic-Line Series *All varieties* by Thunderstone ... *New*	CALL	**CALL**
Microsoft LISP *Common LISP*	250	**189**
PLA microProlog by Programming Logic Associates	250	**219**
with APES	450	**399**
PLA Professional microProlog by Programming Logic	395	**349**
with APES	695	**599**
QNIAL *Combines APL with LISP* by NIAL Systems	375	**359**
Turbo Prolog Compiler by Borland International	100	**79**

assembly language

	LIST	OURS
8088 Assembler *w/Z-80 Translator* by 2500 AD	100	**89**
ASMLIB *Function Library* by BC Associates ... *New*	149	**129**
Cross Assemblers *Over 25 Varieties* from 2500 AD	CALL	**CALL**
Microsoft Macro Assembler *with utilities*	150	**99**
Turbo EDITASM *Fast Assembler* by Speedware	99	**84**
Visible Computer: 8088 by Software Masters	80	**65**

basic language

	LIST	OURS
BetterBASIC by Summit Software	200	**165**
8087 Math Support	99	**85**
Btrieve Interface	99	**85**
C Interface	CALL	**CALL**
Run-time Module	250	**225**
Microsoft QuickBASIC Compiler ... *New version 2.0*	99	**79**
Professional BASIC by Morgan Computing	99	**79**
8087 Math Support	50	**47**
True Basic *Includes FREE BASICA Translator*	150	**105**
Run-time Module	500	**435**

blaise products

	LIST	OURS
ASYNCH MANAGER *Specify for C or Pascal*	175	**136**
C TOOLS/TOOLS 2 Combination Package	175	**149**
C TOOLS	125	**105**
C TOOLS 2	100	**84**
C TOOLS PLUS *for Lattice C 3+ and Microsoft C 3+* ... *New*	175	**139**
EXEC *Program Chainer*	95	**79**
PASCAL TOOLS/TOOLS 2 *Combination Package*	175	**139**
PASCAL TOOLS	125	**105**
PASCAL TOOLS 2	100	**84**
RUNOFF *Text Formatter* ... *New*	50	**47**
TURBO ASYNCH PLUS *for Turbo Pascal* ... *New Version*	100	**84**
TURBO POWER TOOLS PLUS *for Turbo Pascal* ... *New Version*	100	**84**
VIEW MANAGER with Source *Specify for C or Pascal*	275	**209**

borland products

	LIST	OURS
REFLEX *Data Base System*	150	**119**
REFLEX Workshop ... *New*	70	**59**
REFLEX and REFLEX Workshop *Combo Package* ... *New*	200	**159**
Turbo DATABASE TOOLBOX	70	**54**
Turbo EDITOR TOOLBOX	70	**54**
Turbo GAMEWORKS TOOLBOX	70	**54**
Turbo GRAPHIX TOOLBOX	70	**54**
Turbo LIGHTNING	99	**75**
Turbo PASCAL *with 8087 and BCD*	100	**69**
Turbo Prolog Compiler	100	**79**
Turbo TUTOR *for Turbo PASCAL*	40	**32**
Word Wizard ... *New*	70	**59**
Word Wizard and Turbo Lightning *Combo Package* ... *New*	150	**129**

c compilers

	LIST	OURS
C-86 Optimizing Compiler	395	**289**
Datalight C Compiler *Small Memory Model*	60	**49**
Datalight Developer's Kit *with Large Memory Model*	99	**79**
DeSmet C *with Source Debugger*	159	**145**
DeSmet C *with Debugger & Large Case Option* ... *New*	209	**193**
Eco-C *Complete Development System* by Ecosoft	125	**89**
Lattice C Compiler from Lattice ... *See Lattice Section*	500	**299**
Let's C Compiler by Mark Williams	75	**59**
with csd Source Level Debugger	150	**118**
Microsoft C Compiler *with CodeView* ... *New version 4.0*	450	**299**
MWC-86 by Mark Williams	495	**299**
Wizard C Compiler *Includes Lint* by Wizard Systems	450	**369**

c interpreters

	LIST	OURS
C-terp by Gimpel Software *Specify compiler interface*	300	**239**
Instant C by Rational Systems ... *New version*	500	**379**
Introducing C by Computer Innovations	125	**105**
Run/C by Age of Reason	150	**99**
Run/C Professional by Age of Reason	250	**184**

c utilities

Also refer to Blaise, Lattice, Microsoft, Phoenix, Polytron, SoftCraft and Xenix System V sections.

	LIST	OURS
APT Application Programmer's Toolkit by Shaw American	395	**339**
Basic C Library by C Source	175	**135**
C Essentials by Essential Software	100	**85**
C to dBase *ISAM Manager* by Computer Innovations	150	**139**
c-tree *ISAM File Manager with source* by FairCom	395	**329**
C Utility Library by Essential Software	185	**139**
C Windows by Syscom	100	**89**
C Wings by Syscom	50	**45**
CI Probe *Source Level Debugger* ... *Limited Quantity Sale*	225	**159**
CI RomPac *for C-86* by Computer Innovations	195	**149**
dBx *dBase to C Translator* by Desktop AI ... *New*	350	**329**
dbQUERY by Raima ... *New*	CALL	**CALL**
dbVISTA *Single-User DBMS* by Raima	195	**159**
with Source Code	495	**429**
dbVISTA *Multi-User DBMS* by Raima	495	**429**
with Source Code	990	**849**
Entelekon Combo Package *All 3 items below*	200	**175**
C Function Library	130	**115**
C Windows	130	**115**
Superfonts for C	50	**45**
Essential Graphics *No Royalties* by Essential Software	250	**219**
Flash-up Windows by Software Bottling of NY	75	**69**
GraphiC *Mono version 2.2* by Scientific Endeavors	280	**219**
GraphiC *Color version 3.0* by Scientific Endeavors	350	**299**
The Greenleaf Functions by Greenleaf Software	185	**135**
Greenleaf Comm Library by Greenleaf Software	185	**135**
The HAMMER by OES Systems	195	**175**
MetaWINDOWS by Metagraphics	185	**139**
MetaWINDOWS/Plus by Metagraphics	235	**199**
Multi-Halo *with Royalties* by Media Cybernetics	300	**219**
On-line Help from Opt-Tech Data Processing	149	**119**
PANEL by Roundhill *Library Source Also Available*	295	**229**
PC Lint by Gimpel Software ... *New Version*	139	**109**
Scientific Subroutine Library for C by Peerless	175	**139**
Vitamin C by Creative Programming	150	**139**
VC Screen *Interactive Forms Designer*	99	**85**
Zview *with Free Updates* by Data Mgmt Consultants	245	**199**

cobol language

	LIST	OURS
Micro Focus COBOL Workbench	4000	**3599**
Micro Focus Level II COBOL	1500	**CALL**
COGRAPHICS ... *New*	250	**219**
COMATH	200	**169**
FORMS-2	300	**269**
Level II Animator	900	**CALL**
Level II SOURCEWRITER	2000	**CALL**
Micro Focus Level II COBOL *For Novell NetWare*	2000	**1799**
Micro Focus Micro/SPF	175	**159**
Micro Focus Professional COBOL	3000	**2395**
Multi-user Runtime *for PC Network*	500	**449**
Microsoft COBOL ... *See Microsoft Section*	700	**495**
OPT-Tech Sort *Also Sorts Btrieve Files*	149	**119**
Realia COBOL	995	**839**
RM/COBOL by Ryan-McFarland	950	**675**
RM/COBOL 8X *ANSI 85 COBOL* by Ryan-McFarland	1250	**995**

debuggers & profilers

	LIST	OURS
Advanced Trace-86 *with ASM Interpreter* by Morgan	175	**139**
CI Probe *Source Level Debugger* ... *Limited Quantity Sale*	225	**159**
Codesifter *Execution Profiler* by David Smith ... *New*	119	**99**
Codesmith-86 *Debugger* by Visual Age	145	**109**
Periscope I *w/Board & Switch* by Data Base Decisions	295	**249**
Periscope II *w/NMI Breakout Switch Only*	145	**115**
Periscope II-X *Software only* ... *Special Price thru August*	115	**85**

forth language

	LIST	OURS
CFORTH *Native Code Application Compiler* by LMI	300	**239**
LMI Forth/83 Metacompiler *Specify Target Processor*	750	**599**
PC/Forth by Laboratory Microsystems	150	**119**
PC/Forth+ by Laboratory Microsystems	250	**209**

fortran language

	LIST	OURS
ACS Time Series by Alpha Computer Service	495	**429**
Btrieve *ISAM File Manager* ... *See SoftCraft Section*	250	**195**
50 MORE: FORTRAN by Peerless Engineering	125	**99**
For-Winds by Alpha Computer Service	90	**79**
Forlib-Plus by Alpha Computer Service	70	**55**
FORTRAN Addenda by Impulse Engineering ... *New*	95	**89**
FORTRAN Addendum by Impulse Engineering ... *New*	165	**149**
Grafmatic or Plotmatic by Microcompatibles ... *New*	135	**119**
Grafmatic and Plotmatic by Microcompatibles ... *New*	240	**219**
Microsoft Fortran	350	**215**
Multi-Halo *with Royalties* by Media Cybernetics	300	**219**
PANEL *Screen Designer* by Roundhill	295	**229**
RM/Fortran by Ryan-McFarland	595	**394**
Scientific Subroutine Library by Peerless	175	**139**
The Statistician by Alpha Computer Service	295	**259**
Strings & Things by Alpha Computer Service	70	**55**

Circle **no. 129** on reader service card.

lattice products

	LIST	OURS
Lattice C Compiler from Lattice	500	299
with Library Source Code	900	549
C Cross Reference Generator	50	39
with Source Code	200	159
C-Food Smorgasbord *Function Library*	150	99
with Source Code	300	195
C-Sprite *Source Level Debugger*	175	139
Curses *Screen Manager*	125	98
with Source Code	250	194
dBC *dBase File Manager for C*	250	194
with Source Code	500	388
LMK *Make Facility*	195	149
RPG II Compiler *No Royalties*	750	635
SecretDisk *File Encryption Utility*	120	95
SideTalk *Resident Communications*	120	95
Text Mgmt Utilities *(GREP,DIFF,ED,WC,Extract,Build)*	120	95
TopView Toolbasket *Function Library*	250	194
with Source Code	500	388
Z-80 C Cross Compiler	500	388
with Library Source Code	1000	789

logitech products

	LIST	OURS
Logimouse *New*	99	89
Logimouse+ *New*	119	105
Logimouse+ with Paint *New*	169	149
MODULA-2/86 Compiler	89	64
with 8087	129	104
with 512K	189	148
MODULA-2 Editor	59	49
MODULA-2 Runtime Debugger	69	59
MODULA-2 Source Package	179	155
MODULA-2 Translator *TurboPascal to MODULA-2* *New*	49	45
MODULA-2 Utilities Package	49	45
MODULA-2 Window Package *New*	49	45

microsoft products

	LIST	OURS
Microsoft BASIC Interpreter for Xenix	350	279
Microsoft C Compiler *with CodeView* *New version 4.0*	450	299
Microsoft COBOL Compiler	700	495
for Xenix	995	795
Microsoft COBOL Tools *with COBOL Source Debugger*	350	209
for Xenix	450	359
Microsoft Fortran Compiler	350	215
for Xenix	695	CALL
Microsoft LISP *Common LISP*	250	189
Microsoft Macro Assembler *with utilities*	150	99
Microsoft Mouse *Bus Version*	175	149
Microsoft Mouse *Serial Version*	195	159
Microsoft muMath *Includes muSIMP*	300	195
Microsoft Pascal Compiler	300	195
for Xenix	695	CALL
Microsoft QuickBASIC Compiler *New version 2.0*	99	79
Microsoft Sort	195	149
Microsoft Windows	99	74
Microsoft Windows Developer's Kit	500	395

other languages

	LIST	OURS
CCS MUMPS *Single-User version* by MGlobal *New*	60	55
CCS MUMPS *Multi-User version* by MGlobal *New*	450	379
Janus/ADA C Pack by R&R Software	95	89
Janus/ADA D Pack by R&R Software	900	795
Methods *Smalltalk* by Digitalk	79	69
Smalltalk/V by Digitalk *New*	99	89
SNOBOL4+ by Catspaw *New*	95	84

other products

	LIST	OURS
Dan Bricklin's Demo Program by Software Garden	75	65
FASTBACK *Backup Utility* by 5th Generation Systems	179	159
Interactive EASYFLOW by Haventree Software	150	129
Quilt Computing Combo Package *Both Items Below* *New*	199	169
QMake *Program Rebuild Utility* *New*	99	84
SRMS *Software Revision Mgmt System*	125	109
Source Print by Aldebaran Laboratories	139	119
Taskview by Sunny Hill Software *New*	70	59

phoenix products

	LIST	OURS
Pfantasy Pac *(Pfinish,Pfix+,Plink+,Pmaker,Pmate,Ptel)*	1295	949
Pfinish *Performance Analyzer* *New Version*	395	259
Pfix-86 Plus *Symbolic Debugger*	395	259
PforCe *Comprehensive C Function Library*	395	289
Plink-86 *Overlay Linker*	395	259
Plink-86 Plus *Enhanced Overlay Linker*	495	359
Pmate *Macro Text Editor*	195	149
Pre-C *Lint Utility*	295	195
Ptel *Binary File Transfer Program*	195	149

polytron products

	LIST	OURS
Polytron C Beautifier	49	45
Polytron C Library I	99	79
Polytron PowerCom *Communications*	179	139
PolyLibrarian *Library Manager*	99	79
PolyLibrarian II *Library Manager*	149	119
PolyMake *UNIX-like Make Facility*	99	79
PolyOverlay *Overlay Optimizer*	99	79
PolyWindows Products *All Varieties*	CALL	CALL
PolyXREF *Complete Cross Reference Utility*	219	179
PolyXREF *Support for one language only*	129	109
PVCS *Polytron Version Control System*	395	329
PVMFM *Polytron Virtual Memory File Manager*	199	149

softcraft products

	LIST	OURS
Btrieve *ISAM File Manager with No Royalties*	250	195
Xtrieve *Query Utility for Btrieve*	195	165
Rtrieve *Report Generator for Xtrieve*	85	75
Btrieve/N *for Networks*	595	465
Xtrieve/N *Query Utility for Btrieve/N*	395	299
Rtrieve/N *Report Generator for Xtrieve/N*	175	159

text editors

	LIST	OURS
Brief from Solution Systems	195	CALL
Epsilon *Multi-tasking Emacs-like editor* by Lugaru	195	165
FirsTime for Turbo by Spruce Technology	75	69
KEDIT *Xedit-like editor* by Mansfield Software Group	125	109
PC/VI by Custom Software Systems *New*	149	129
Personal REXX by Mansfield Software Group *New*	125	109
SPF/PC by Command Technology Corp	195	165
Vedit by CompuView	150	115
Vedit Plus by CompuView	225	180
XTC *Text Editor with source* by Wendin	99	84

turbo pascal utilities

Also refer to Blaise, Borland and SoftCraft sections.

	LIST	OURS
ALICE *Turbo Pascal Interpreter* by Software Channels	95	69
FirsTime for Turbo by Spruce Technology	75	69
Flash-up Windows by Software Bottling of NY	75	69
Multi-Halo *with Royalties* by Media Cybernetics	CALL	CALL
On-line Help from Opt-Tech Data Processing	149	119
Screen Sculptor by Software Bottling of NY	125	95
Turbo EXTENDER by TurboPower Software	85	69
Turbo Professional by Sunny Hill Software	70	49
TurboPower Utilities by TurboPower Software	95	84
TurboWINDOW by MetaGraphics	80	69

wendin products

	LIST	OURS
Operating System Toolbox *Build your own OS*	99	84
PCUNIX *Operating System*	99	84
PCVMS *Operating System Similar to VAX/VMS*	99	84
XTC *Text Editor with Pascal Source Code*	99	84

xenix system v

	LIST	OURS
Complete Xenix System by SCO *All 3 items below*	1295	1099
Xenix Development System	595	529
Xenix Operating System *Specify XT or AT*	595	529
Xenix Text Processing Package	195	169

xenix languages and utilities

	LIST	OURS
APL*PLUS/UNX System *For AT Xenix* by STSC	995	795
Btrieve *ISAM File Manager* by SoftCraft	595	465
c-tree *ISAM File Manager with Source* by FairCom	395	329
dBx *dBase to C Translator w/source* by Desktop AI *New*	550	499
dbVISTA *Single and Multi User versions* by Raima	CALL	CALL
Informix by Relational Database Systems	995	839
Informix4GL by Relational Database Systems *New*	CALL	CALL
InformixSQL by Relational Database Systems *New*	995	839
Lyrix by SCO	595	489
Micro Focus Level II Compact COBOL *For AT*	1000	899
Forms-2	400	359
Level II ANIMATOR	600	539
Microsoft Languages *See Microsoft Section*		
Networks for XENIX by SCO	595	529
PANEL *Screen Designer for AT Xenix* by Roundhill	695	CALL
RM/COBOL by Ryan-McFarland	1250	995
RM/FORTRAN by Ryan-McFarland	750	599
SCO Professional *Complete Lotus clone* by SCO	795	695

VISA

Prices are subject to change without notice.
Ohio customers please add 5% state sales tax.

MasterCard

Hours: 8:30 AM to 8:00 PM EST, Monday through Friday.

U.S.
1-800-336-1166
CANADA
1-800-225-1166
OHIO AND OVERSEAS
1-216-877-3781
CUSTOMER SERVICE **216-877-1110**

programmer's connection

136 SUNNYSIDE ST.
HARTVILLE, OHIO 44632

Circle no. **129** on reader service card.

Listing One

(Listing continued, text begins on page 14.)

```
        dp->path = 1;                   /* and include the full path name.   */
        dir( dname, dp );

        vects = (char **) dp->dirv;         /* pointer to list of subdirs  */
        count = dp->ndirs;                  /* number of subdirs.          */

        while( --count >= 0 )                   /* visit the subdirs, one */
        {                                       /* at a time. the setbit  */
                setbit( depth,  count );        /* call is used for       */
                prnt( *vects++, count );        /* printing the tree.     */
        }

        if( !others )                           /* If there aren't any    */
                pline( depth, 1 );              /* subdirs in the parent, */
                                                /* output a blank line    */
        del_dir( dp );
        --depth;
}

/* ----------------------------------------------------------------- */

char    *dodot( str )
char    *str;
{
        /* If str has no dots in it, return str, else get the pathname
         * refered to by str (ie. whatever name is indicated by
         * . or .. or ../.. etc.) and return a pointer to that string.
         */

        static char root_name[ DSIZE ];
        char        *p;

        if( !strchr( str, '.' ) )
                return str;

        if( chdir(str)  ||  !getcwd(root_name, DSIZE) )
        {
                fprintf(stderr, "Can't find %s, aborting\n", str );
                exit( 1 );
        }

        for( p = root_name; *p ; p++ )          /* Map the name from DOS  */
        {                                       /* style to UNIX style by */
                if( *p == '\\' )                /* mapping upper to lower */
                        *p = '/';               /* case and changing \ to */
                else                            /* /                      */
                        *p = tolower( *p );
        }

        chdir( Startdir );                      /* Restore the original   */
        return root_name ;                      /* working directory      */
}

/* ----------------------------------------------------------------- */

doargs( argc, argv )
char    **argv;
{
        /*  Does several things. First, it shifts all the arguments down
         *  one notch, overwriting the original argv[0]. Next, it
         *  puts a NULL into argv[argc-1], finally it processes (and
         *  removes from argv) all command line switches. Switch processing
         *  stops after a -e is encountered (but the compression continues).
         *  Argc, decremented to reflect all this stuff, is returned.
         *
         *  We can't use getargs() in the program because -e is
         *  position dependant.
         */

        register int    nargc;
        register char   **nargv;

#ifdef DEBUG
        char  **v = argv;
        int     c;
#endif

        nargc = 0 ;
        for( nargv = argv++; --argc > 0; argv++ )
        {
                if( **argv != '-' || Args )
                {
                        *nargv++ = *argv ;
                        nargc++;
                }
                else
                {
                        switch( argv[0][1] )
                        {
                        case 'e':
                                Args      = nargv   ;
                                *nargv++ = *++argv ;
                                nargc ++;
                                putenv("CMDLINE=") ;
                                break;

                        case 'f':  Findfile    = &argv[0][2];    break;
                        case 's':  Short_pname = 1;              break;
                        case 'd':  Draw        = 1;              break;
                        default :  usage();
                        }
                }
        }

```


(continued on page 68)

Circle **no. 183** on reader service card.

C CHEST

Listing One

(Listing continued, text begins on page 14.)

```
        *argv = NULL ;  /* Add a NULL as the last entry */

#ifdef DEBUG
        printf("New argv is:\n");

        for( c = nargc; --c >= 0 ; v++)
                printf("<%s>  0x%x\n", *v, *v );

        printf("\nFindfile=<%s>, Short_pname=%d, Draw=%d, *Args=0x%x\n",
                                        Findfile, Short_pname, Draw, *Args);
#endif

        return nargc ;
}

/* ---------------------------------------------------------------------- */

#define E(x) fprintf(stderr, "%s\n", x )
usage()
{
     E( "\nUsage is: dtree root [-s] [-d] [-f<name>] [-e arg arg arg]\n");
     E("-e        Execute rest of cmd line from each directory" );
     E("-f<name>  Find file called <name>" );
     E("-s        Use short path names" );
     E("-d        Draw directory tree"        );
     E("\nEach switch must be in its own argument (-sd is illegal,");
     E("you must say -s -d). If -f and -e are both specified, the command");
     E("is only executed if the indicated file is found.");
     exit(1);
}

wusage()
{
    E( "\nUsage is: whereis <filename>\n");
    E( "Only one file name is permitted, though wildcards are recognized");
    E( "by whereis itself, so you must escape these from the shell as in:");
    E( "\twhereis \"*.c\"    or     whereis \\*.c" );
    exit(1);
}

/* ---------------------------------------------------------------------- */

onintr()
{
                                        /* Called when a ^C is encountered:     */
        chdir( Startdir );              /* Get back to starting directory       */
        exit(0);                        /* before exiting.                      */
}

/* ---------------------------------------------------------------------- */

main( argc, argv )
char    **argv;
{
        /*      If the program is invoked under the name "whereis" it
         *      treats the command line: whereis <fname>
         *      as if you had said:      dtree / -f<fname>
         */

        reargv( &argc, &argv ); /* Redo arg list if running under shell */

        if( !strcmp(*argv, "whereis") )
        {
                if( argc != 2 || argv[1][0] == '-' )
                        wusage();

                Findfile = argv[1];     /* Search for a file.           */
                argc = 0;               /* Force search to begin at /   */
        }
        else
        {
                argc = doargs( argc, argv );
                                                /*  argv[0] [1] [2]  ...  */
                if( Args &&  argc < 2 )         /* pathname cmd args ...  */
                        usage();
        }


        Cset = isatty(fileno(stdout)) ? Graph_chars : Norm_chars ;

        if( !getcwd(Startdir, DSIZE) )
        {
                fprintf(stderr, "Can't save current directory, aborting\n");
                exit( 1 );
        }

        signal( SIGINT, onintr );
        prnt( (argc < 1 || argv == Args) ? "/" : dodot(argv[0]), 0 );
        chdir( Startdir );

        exit(0);
}
```

End Listing One

Listing Two

```
                        Listing 2 -- fix.c
-----------------------------------------------------------------------
#include <stdio.h>
#include <fcntl.h>
#include <types.h>
#include <stat.h>

extern  char    *strrchr();
```

(continued on page 70)

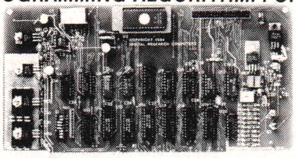
ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A COPY OF THIS WARRANTY IS AVAILABLE FREE, ON REQUEST.

DIGITAL RESEARCH COMPUTERS

(214) 225-2309

S100 EPROM PROGRAMMER

OUR NEWEST DESIGN, FOR FAST EFFICIENT PROGRAMMING OF THE MOST POPULAR EPROM'S ON YOUR S100 MACHINE. COMES WITH MENU DRIVEN SOFTWARE THAT RUNS UNDER CP/M 2.2 (8 INCH). PC BOARD SET CONSISTS OF (S100) MAIN LOGIC BOARD REMOTE PROGRAMMING CARD AND SIX PERSONALITY MINI BOARDS FOR 2716, 2532, 2732, 2732A, 2764, AND 27128. SOLD AS BARE PC BOARD SET ONLY WITH FULL DOC. SOFTWARE FEATURES "FAST" PROGRAMMING ALGORITHM. FOR Z80 BASED SYSTEMS.

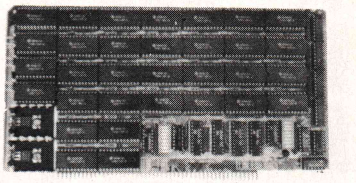


PC BOARD SET, FULL DOCUMENTATION, 8 IN. DISKETTE WITH SOFTWARE.

NEW! $69.95

64K S100 STATIC RAM

$99.00 KIT



LOW POWER!

150 NS ADD $10

BLANK PC BOARD WITH DOCUMENTATION $49.95

SUPPORT ICs + CAPS $17.50

FULL SOCKET SET $14.50

FULLY SUPPORTS THE NEW IEEE 696 S100 STANDARD (AS PROPOSED)

ASSEMBLED AND TESTED ADD $50

FEATURES: *PRICE CUT!*

- Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
- Fully supports IEEE 696 24 BIT Extended Addressing.
- 64K draws only approximately 500 MA.
- 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
- SUPPORTS PHANTOM (BOTH LOWER 32K AND ENTIRE BOARD).
- 2716 EPROMs may be installed in any of top 48K.
- Any of the top 8K (E000 H AND ABOVE) may be disabled to provide windows to eliminate any possible conflicts with your system monitor, disk controller, etc.
- Perfect for small systems since BOTH RAM and EPROM may co-exist on the same board.
- BOARD may be partially populated as 56K.

128K S100 STATIC RAM/EPROM BOARD

JUST OUT! USES POPULAR 8K X 8 STATIC RAMS (6264) OR 2764 EPROMS. FOR 8 OR 16 BIT DATA TRANSFERS! IEEE 696 STANDARD. LOW POWER. KITS ARE FULLY SOCKETED. FULL DOC AND SCHEMATICS INCLUDED. 24 BIT ADDRESSING.

NEW!

$59.95 BARE PC BOARD

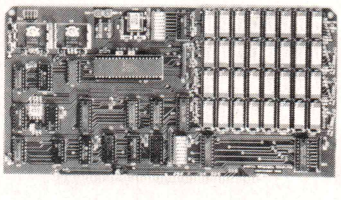
$219.00 128K RAM KIT

$139.00 128 EPROM KIT

256K S-100 SOLID STATE DISK SIMULATOR!

WE CALL THIS BOARD THE "LIGHT-SPEED-100" BECAUSE IT OFFERS AN ASTOUNDING INCREASE IN YOUR COMPUTER'S PERFORMANCE WHEN COMPARED TO A MECHANICAL FLOPPY DISK DRIVE.

PRICE CUT!



FEATURES:

- 256K on board, using + 5V 64K DRAMS.
- Uses new Intel 8203-1 LSI Memory Controller.
- Requires only 4 Dip Switch Selectable I/O Ports.
- Runs on 8080 or Z80 S100 machines.
- Up to 8 LS-100 boards can be run together for 2 Meg. of On Line Solid State Disk Storage.
- Provisions for Battery back-up.
- Software to mate the LS-100 to your CP/M* 2.2 DOS is supplied.
- The LS-100 provides an increase in speed of up to 7 to 10 times on Disk Intensive Software.
- Compare our price! You could pay up to 3 times as much for similar boards.

BLANK PCB (WITH CP/M* 2.2 PATCHES AND INSTALL PROGRAM ON DISKETTE) $49.95 (8203-1 INTEL $29.95)

(ADD $50 FOR A&T) $129.00 #LS-100 (FULL 256K KIT)

PANASONIC

Green Screen - Video Monitors

25 MHZ. TYPICAL BANDWIDTH!!!

Brand New In The Box! 9-Inch Screen

#K-904B1 (Chassis #Y08A) Open Frame Style

$29.95 EA.

GROUP SPECIAL: 4 for $99.00 WITH DATA & SCHEMATIC

(USA SHIPPING: $3. PER UNIT. CANADA: $7. PER UNIT)

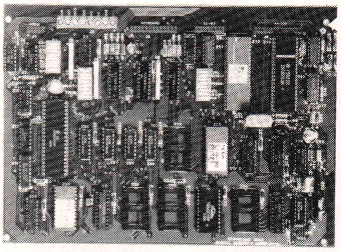
COMPUTER MANUFACTURER'S EXCESS. STILL IN ORIGINAL PANASONIC BOXES. THE CRT TUBE ALONE WOULD COST MORE THAN OUR PRICE FOR THE COMPLETE UNIT. FOR SPLIT VIDEO (TTL INPUTS) OPERATION, NOT COMPOSITE VIDEO. OPERATES FROM 12VDC AT 1 AMP. VERTICAL INPUT IS 49 TO 61 HZ. HORIZONTAL INPUT: 15,750 HZ ± 500 HZ. RESOLUTION IS 800 LINES AT CENTER 650 LINES AT CORNERS.

ZRT-80
CRT TERMINAL BOARD!

A LOW COST Z-80 BASED SINGLE BOARD THAT ONLY NEEDS AN ASCII KEYBOARD, POWER SUPPLY, AND VIDEO MONITOR TO MAKE A COMPLETE CRT TERMINAL. USE AS A COMPUTER CONSOLE, OR WITH A MODEM FOR USE WITH ANY OF THE PHONE-LINE COMPUTER SERVICES.

FEATURES:

- Uses a Z80A and 6845 CRT Controller for powerful video capabilities.
- RS232 at 16 BAUD Rates from 75 to 19,200.
- 24 x 80 standard format (60 Hz).
- Optional formats from 24 x 80 (50 Hz) to 64 lines x 96 characters (60 Hz).
- Higher density formats require up to 3 additional 2K x 8 6116 RAMS.
- Uses N.S. INS 8250 BAUD Rate Gen. and USART combo IC.
- 3 Terminal Emulation Modes which are Dip Switch selectable. These include the LSI-ADM3A, the Heath H-19, and the Beehive.
- Composite or Split Video.
- Any polarity of video or sync.
- Inverse Video Capability.
- Small Size: 6.5 x 9 inches.
- Upper & lower case with descenders.
- 7 x 9 Character Matrix.
- Requires Par. ASCII keyboard.



$89.95 #ZRT-80

A&T ADD $50

(COMPLETE KIT, 2K VIDEO RAM)

FOR 8 IN. SOURCE DISK (CP/M COMPATIBLE) ADD $10

BLANK PCB WITH 2716 CHAR. ROM. 2732 MON. ROM $49.95

SOURCE DISKETTE - ADD $10

SET OF 2 CRYSTALS - ADD $7.50

THE NEW 65/9028 VT ANSI VIDEO TERMINAL BOARD!

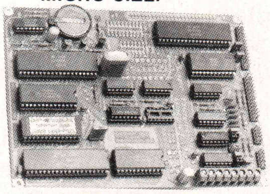
★ FROM LINGER ENTERPRISES ★

A second generation, low cost, high performance, mini sized, single board for making your own RS232 Video Terminal. Use as a computer console or with a MODEM for hook up to any of the telephone-line computer services.

FEATURES:

- Uses the new SMC 9028 Video Controller Chip coupled with a 6502A CPU.
- RS-232 at 16 Baud Rates from 50 to 19,200
- On board printer port!
- 24 X 80 format (50/60 Hz).
- For 15,750 Hz (Horiz.) monitors.
- 3 Terminal Modes: H-19, ADM3A, and ANSI X 3.64-1979
- Wide and thin-line graphics.
- White characters on black background or reversed.
- Character Attributes: De-Inten, Inverse, Underline and Blank.
- Low Power: 5VDC @ .7A, ± 12VDC @ 20MA.
- Mini size: 6.5 X 5 inches.
- Composite or split video.
- 5 X 8 Dot Matrix characters (U/L case) with descenders.
- Answer back capability.
- Battery backed up status memory.
- For ASCII parallel keyboard.

MICRO SIZE!



$99.95 (Full Kit)

ADD $40 FOR A&T

SOURCE DISKETTE: PC/XT FORMAT 5¼ IN. $15

OPTIONAL EPROM FOR PC/XT STYLE SERIAL KEYBOARD: $15

Digital Research Computers

P.O. BOX 381450 • DUNCANVILLE, TX 75138 • (214) 225-2309

TERMS: Add $3.00 postage. Orders under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5-1/8% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50 add 85¢ for insurance.

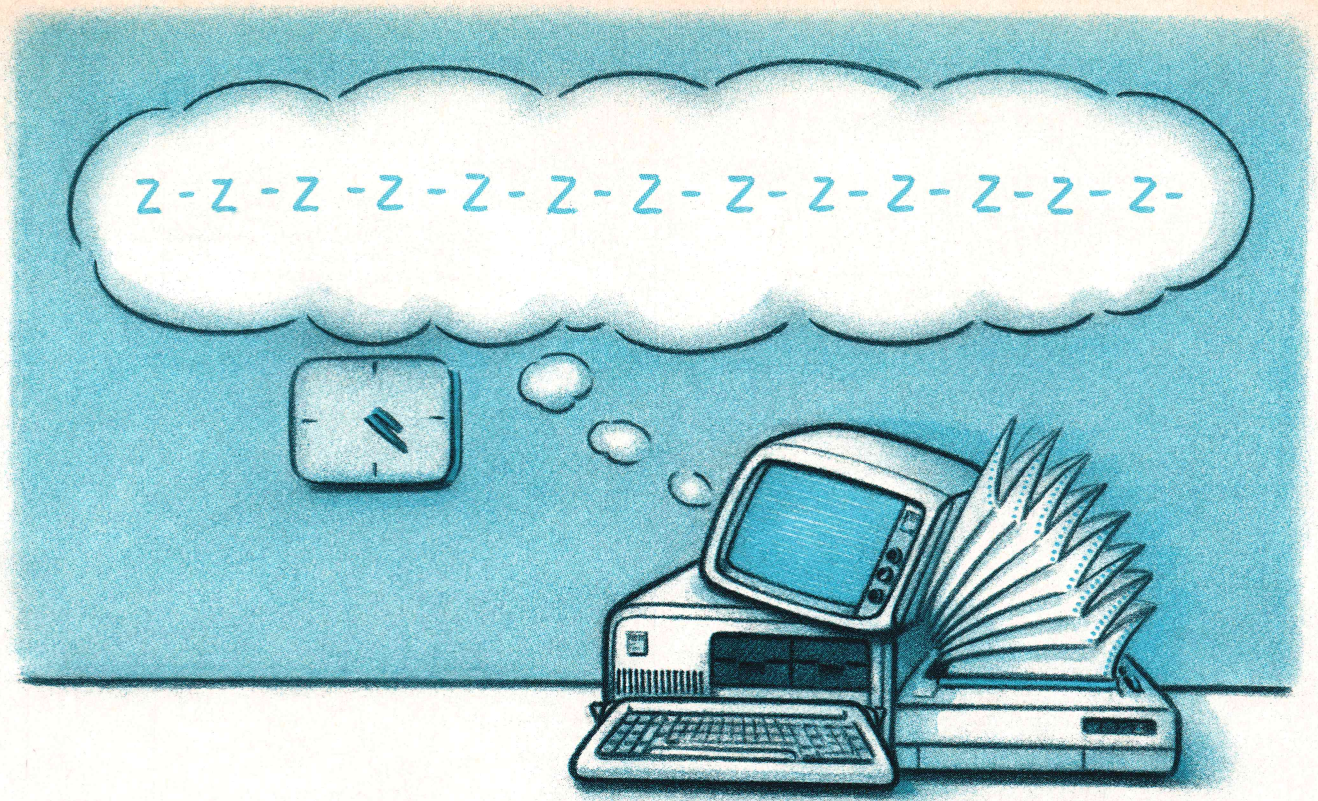
*TM OF DIGITAL RESEARCH INC. (CALIF.)

WE ARE NOT ASSOCIATED WITH DIGITAL RESEARCH INC. (CALIF.) THE SUPPLIERS OF CPM SOFTWARE

Listing Two (Listing continued, text begins on page 14.)

```
#define BSIZE    (10 * 1024)                              /* Buffer size        */
#define CTL_Z    0x1a                                /* EOF marker        */
#define SMODE    (O_RDONLY | O_BINARY )              /* read & write modes */
#define DMODE    (O_WRONLY | O_BINARY | O_TRUNC | O_CREAT )

/*------------------------------------------------------------------------*/

char    *bak( name )
char    *name;
{
        /* Strips extension from name and adds .bak extension, returning  */
        /* a pointer to the modified name. The original name is untouched */

        static char buf[128], *p;

        strncpy( buf, name, 128-5 );

        if( p = strrchr(buf, '.') )
                strcpy( p+1, "bak" );
        else
                strcat( buf, ".bak" );

        return buf;
}

/*------------------------------------------------------------------------*/

usage()
{
        fprintf(stderr, "Usage:  fix file [file...]\n\n");
        fprintf(stderr, "Removes trailing ^Z's from files.\n");
        exit( 1 );
}

/*------------------------------------------------------------------------*/

main(argc, argv)
char    **argv;
{
        static   char   buf[BSIZE];
        static   char   *srcname;
        char            *p;
        register int    got;                /* # bytes got from read     */
        register int    src, dest;          /* File handles              */

        ctlc();                             /* Fix ^C Interrupt handling */
        reargv( &argc, &argv );             /* Remake argv from CMDLINE  */

        if( argc < 2 || argv[1][0] == '-' )
                usage();

        for( ++argv, --argc;  --argc >= 0;  ++argv )
        {
                srcname = bak( *argv   );   /* srcname = xxx.bak          */
                unlink( srcname        );   /* delete  xxx.bak            */
                rename( srcname, *argv );   /* rename  xxx.yyy to xxx.bak */

                printf("Fixing %-20s (creating %s)\n", *argv, srcname );

                if( (src = open(srcname, SMODE)) == -1 )
                {
                        perror( srcname );
                        continue;
                }
                if( (dest = open( *argv, DMODE, S_IWRITE | S_IREAD)) == -1 )
                {
                        perror( *argv );
                        continue;
                }

                while( got = read(src, buf, BSIZE) )
                {
                        if( got == -1 )
                        {
                                perror( srcname );
                                break;
                        }

                        for( p = buf; --got >= 0  &&  *p != CTL_Z ; p++ )
                                ;

                        got = p - buf ;    /* got = distance to ^Z */


                        if( write(dest, buf, got) != got )
                        {
                                perror( *argv );
                                break;
                        }

                        if( *p == CTL_Z )
                                break;
                }

                close( src  );
                close( dest );
        }

        exit( 0 );
}
```

End Listings



QNX: With any other OS, your personal computer is asleep at the switch.

If real-time performance is the key to the next generation of small systems, task-switching is the key to real-time performance.

QNX task-switching has been measured using standard Intel benchmarks. Here are the results:

OS	Computer	Processor	Task Switches/ Sec
QNX™	IBM AT	8Mhz 80286	2800
QNX™	IBM PC	5Mhz 8088	787
XENIX™	Intel-286	5Mhz 80286	203
UNIX™	CODATA	8Mhz 68000	187
XENIX™	ALTOS	5Mhz 8086	96
UNIX™	FORTUNE	6Mhz 68000	95

The margin by which QNX outperforms UNIX-based systems is not accidental. QNX architecture is unique among multi-tasking small computer operating systems because it is modular, not monolithic. On the PC, this distinction is decisive: UNIX system overhead and processing demands sap any computer smaller than an AT.

Because QNX was developed specifically for smaller computers, not "kluged" from an OS written for larger computers, its structural advantages are apparent. QNX was the first multi-tasking, multi-user OS for the IBM PC (1982), the first again for the AT (1984) and the first networking OS for the PC and the AT (1984). For the forseeable future, QNX will remain the only real-time multi-user, multi-tasking OS for the PC, the AT and compatibles.

As a networking OS, only QNX provides integrated performance at the fundamental level of inter-task communications. The tasks need not be on the same machine: the messaging is network-wide.

A task responsible for handling an A/D converter on one machine can provide data and accept requests from tasks on any other machine. For this reason only QNX can implement true distributed processing, distributed file systems and distributed devices, with no need for a central file server.

Since QNX is modular, programmers can easily adapt and enhance the OS itself by adding their own system task. PC-DOS 2.1 or 3.1™ can run as one of QNX's many tasks.

With over 20,000 QNX systems installed worldwide, QNX is the undisputed standard for real-time microprocessor applications.

Multi-User	10 serial terminals per PC, AT.	**C Compiler**	Standard Kernighan and Ritchie.
Multi-Tasking	40 (64) tasks per PC (AT).	**Flexibility**	Single PC, networked PC's, Single PC with terminals, Networked PC's with terminals. No central servers. Full sharing of disks, devices and CPU's.
Networking	2.5 Megabit token ring. 255 PC's and/or AT's per network. 10,000 tasks per network. thousands of users per network.		
Real Time	2800 task switches/sec (AT).	**PC-DOS**	PC-DOS runs as a QNX task.
Message Passing	Fast intertask communication between tasks on any machine.	**Cost**	From US $450. Runtime pricing available.

For further information or a free demonstration diskette, please telephone **(613) 726-1893.**

The only multi-user, multi-tasking networking operating system QNX for the IBM PC, AT and compatibles. By Quantum.

Quantum Software Systems Ltd., Moodie Drive High Tech Park, 215 Stafford Road, Unit 104, Ottawa, Ontario, Canada K2H 9C1

Circle **no. 158** on reader service card.

Listing One (Text begins on page 24.)

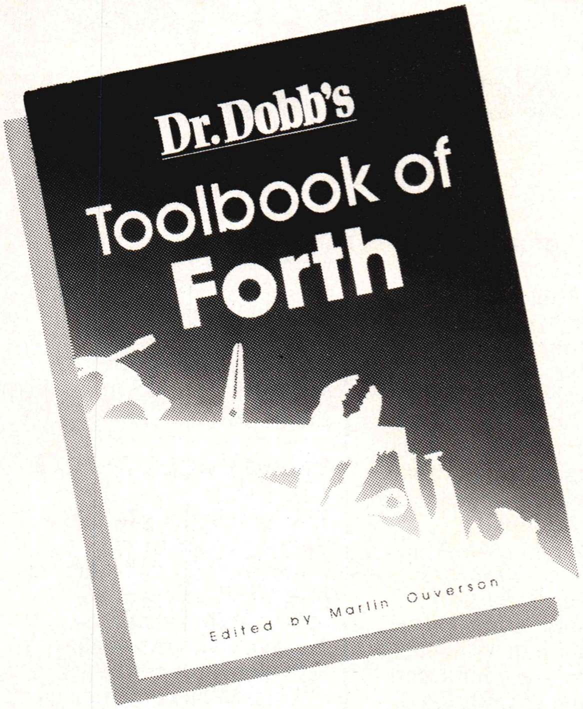
```
/* SPLINE.C - Interpolate Smooth Curve
 *
 * Version 2.00            December 25th, 1985
 *
 * Modifications:
 *
 *   V1.00 (85/11/01)   - beta test release
 *   V2.00 (85/12/25)   - general revision
 *
 * Copyright 1985:      Ian Ashdown
 *                      byHeart Software
 *                      620 Ballantree Road
 *                      West Vancouver, B.C.
 *                      Canada V7S 1W3
 *
 * This program may be copied for personal, non-commercial use
 * only, provided that the above copyright notice is included in
 * all copies of the source code. Copying for any other use
 * without previously obtaining the written permission of the
 * author is prohibited.
 *
 * Synopsis:     SPLINE [option] ...
 *
 * Description: SPLINE takes pairs of numbers from the standard
 *              input as abscissae and ordinates of a function.
 *              (A minimum of four pairs is required.) It
 *              produces a similar set, which is approximately
 *              equally spaced and includes the input set, on the
 *              standard output. The cubic spline output (R.W.
 *              Hamming, "Numerical Methods for Scientists and
 *              Engineers", 2nd ed. 349ff) has two continuous
 *              derivatives and sufficiently many points to look
 *              smooth when plotted.
 *
 *              The following options are recognized, each as a
 *              separate argument:
 *
 *              -a  Supply abscissae automatically (they are
 *                  missing from the input); spacing is given by
 *                  the next argument or is assumed to be 1 if
 *                  next argument is not a number.
 *
 *              -k  The constant "k" is used in the boundary
 *                  value computation
 *
 *                      y " = ky " , y " = ky "
 *                       0      1    n      n-1
 *
 *                  is set by the next argument. By default,
 *                  k = 0. A value of k = 0.5 often results in a
 *                  smoother curve at the endpoints than the
 *                  default value. Negative values for k are not
 *                  allowed. Cannot be used with -p option.
 *
 *              -n  Next argument (which must be an integer)
 *                  specifies the number of intervals that are to
 *                  occur between the lower and upper "x" limits.
 *                  If -n option is not given, default spacing is
 *                  100 intervals.
 *
 *              -p  Make output periodic, i.e. match derivatives
 *                  at ends. First and last input values must
 *                  agree. Cannot be used with -k option.
 *
 *              -x  Next 1 (or 2) arguments are lower (and upper)
 *                  "x" limits. Normally these limits are
 *                  calculated from the data. Automatic abscissae
 *                  start at lower limit (default 0). If either
 *                  argument is outside of the range of
 *                  abscissae, it is ignored.
 *
 * Diagnostics: When data is not strictly monotone in "x", SPLINE
 *              reproduces the input without interpolating extra
 *              points.
 *
 * Bugs:        A limit of 1000 input points is silently
 *              enforced.
 *
 *              The -n option has not been implemented in
 *              accordance with the "UNIX Programmer's Manual"
 *              specification. This was done to avoid ambiguities
 *              when the -n option follows the -x option with one
 *              argument.
 *
 *              At certain negative values for the -k option (for
 *              example, k equals -4.0), the curve becomes
 *              discontinuous. The -k option value has thus been
 *              arbitrarily constrained to be greater than or
 *              equal to zero.
 *
 * Credits:     The above description is a reworded and expanded
 *              version of that appearing in the "UNIX Programmer's
 *              Manual", copyright 1979, 1983 Bell Laboratories.
 */

/*** Definitions ***/

#define FALSE      0
#define TRUE       1
#define MAX_SIZE  1000   /* Input point array limit */

#define ILL_ARG    0    /* Error codes */
#define ILL_CMB    1
#define ILL_KVL    2
#define ILL_NVL    3
#define ILL_OPT    4
#define ILL_XVL    5
#define INS_INP    6
#define MIS_KVL    7
#define MIS_NVL    8
#define MIS_XVL    9
#define MIS_YVL   10
#define NMT_ORD   11

#define SQUARE(a) a*a
#define CUBE(a) a*a*a

/*** Typedefs ***/

typedef int BOOL;       /* Boolean flag */

/*** Include Files ***/

#include <stdio.h>
#include <ctype.h>
#include <math.h>

/*** Main Body of Program ***/

int main(argc,argv)
int argc;
char **argv;
{
  int n = 0,
      i,
      j,
      n_val = 0,
      atoi();
  float x[MAX_SIZE],
        y[MAX_SIZE];
  double a_val = 1.0,
         k_val = 0.0,
         x1_val = 0.0,
         x2_val = 0.0,
         x_intvl,
         ix,
         iy,
         d2y[MAX_SIZE],
         h,
         atof(),
         fabs(),
         spl_int();
  char buffer[257],
       *temp,
       *gets();
  BOOL aflag = FALSE,    /* Command-line option flags */
       kflag = FALSE,
       pflag = FALSE,
       x1flag = FALSE,
       x2flag = FALSE,
       is_float();
  void spl_coeff(),
       pspl_coeff(),
       error();

  /* Parse the command line for user-selected options */

  while(--argc)
  {
    temp = *++argv;
    if(*temp != '-')      /* Check for legal option flag */
      error(ILL_OPT,*argv);
    else
      switch(toupper(*++temp))
```

(continued on page 81)

Dr. Dobb's Catalog

Inside:

Dr. Dobb's **Bound Volumes**
Dr. Dobb's **Listings on Disk**
Dr. Dobb's **Z80 Toolbook**
A Complete **C** Toolbox
A **NEW** Version of the
Unix-like Shell &
Utility Package
for MS-DOS



The Toolbook of FORTH

This comprehensive collection of useful Forth programs and tutorials contains exanded and revised versions of *DDJ's* best Forth articles, along with new Forth material. In addition to the practical code and tutorials, you'll glean important insights about the potential of this increasingly popular language from the many in-depth discussion of advanced Forth topics.

You'll find sections on:

Forth—the Language, including "The Forth Philosophy," "Teaching Forth as a First Language" and "Forth-83 and Vocabularies"

Mathematics in FORTH including "Series Expansion in Forth," "FORTH Floating-Point Package," "Signed Integer Division"

Modifications/ Extensions, including "A Proposal for Strings in Forth," "Non-Deterministic Control Words," "Some Forth Coding Standards," "Towards a More Writable Forth Syntax"

Implementing FORTH, including "Forth and the Motorola 68000," "A 68000 Forth Assembler," "A Forth Assembler for the 6502," "Z8000 Forth"

Forth Programs, including "GO in Forth," "Elements of a Forth Data-Base Design," "The Forth Sort," "SEND & RECV," "Interface for a Mouse," "Relocating Loader in Forth," "Forth Decompiler," "Screen-Oriented Editor Revisited," "Evolution of a Video Editor," "H-19 Screen Editor" and "The Conference Tree"

You'll also find Appendices that will help you convert fig-Forth to Forth-83, and tell you how to stay up-to-date on the latest developments and refinements of this popular language.

The screens in this book are also available on disk as Ascii files. Receive *Dr. Dobb's Toolbook of Forth,* along with the software on disk, together for only $39.95.

Dr. Dobb's Toolbook of Forth
Item #030 $22.95

Toolbook with Disk
Item #031 $39.95

Please specify MS/PC DOS, Apple II, Macintosh, or CP/M. For CP/M disks, specify Osborne or 8″ SS/SD.

To Order:

To order any of Dr. Dobb's *products, return the order form at the end of this catalog, or*

CALL TOLL FREE 1-800-528-6050 EXT. 4001

and refer to product item number, title, and disk format.
For customer service questions,

CALL M&T PUBLISHING, INC. 415-366-3600 EXT., 216

M&T BOOKS

Programs by the Pound

Dr. Dobb's Bound Volumes

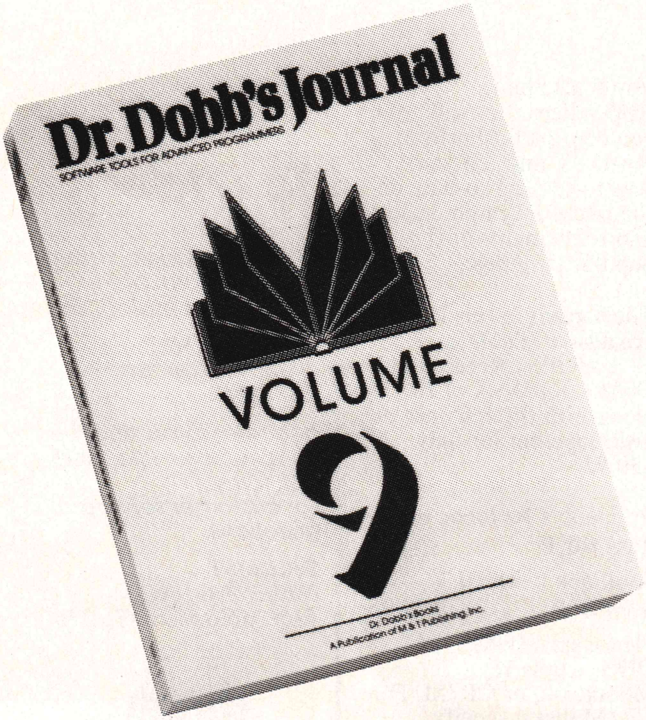
Announcing: Dr. Dobb's Bound Volume 9

Over 1000 pages of listings and text. The entire 1984 editorial contents of *Dr. Dobb's Journal of Software Tools.*

Bound Volume 9: 1984

Item #020B

Shaping things to come. In 1984 new Editor-in-chief Mike Swaine brought his interests in advanced technology to *Dr. Dobb's Journal.* We presented the concepts behind Prolog and published an expert system for weather prediction. We learned Modula-2, and taught Forth to talk to a 68000, to MS DOS, and to the people of China. We examined a new language implementation called Turbo Pascal and extended implementations of C with a preprocessor, a library, Tony Skjellum's tricks, and Allen Holub's Grep. We published two powerful encryption systems, telecommunications protocols, floating-point benchmark results, and an issue devoted to the internals of Unix. And Ray Duncan, Bob Blum, and Dave Cortesi were on hand with their fascinating columns.



Bound Volume 1: 1976

Item #013

The working notes of a technological revolution. Programmers from Defense laboratory systems analysts to kitchen-table entrepreneurs worked for the intrinsic rewards to put development software on the brand-new invention, the microcomputer. Before there was an Apple, *Dr. Dobb's Journal of Tiny Basic Calisthenics and Orthodontia* (subtitle: Running Light without Overbyte) was founded to put a programming language on the machine, and became both chronicler and instrument of the revolution. In this first-year volume: Tiny Basic, the first word on CP/M, notes on building an IMSAI, floating-point and timer routines.

Bound Volume 2: 1977

Item #014

Running light without overbyte. By year two, *Dr. Dobb's* formula was concocted: tough questions and serious technical issues handled with enthusiasm, and wit, scant reverence for the accepted answers. Source code. Tools for programmers. Respect for tight programming. *Dr. Dobb's Journal* readers shared insights on warping the Intel 8080 into a computer CPU, and *Dr. Dobb's* published a complete operating system for the chip. A motley crop of computers and software products were popping up, and *Dr. Dobb's* investigated: the Heath H-8, the KIM-1, the Alpha Micro, MITS Basic, Poly Basic, and Lawrence Livermore Labs Basic. *Dr. Dobb's* introduced Pilot for microcomputers and published tips on doing string handling, high-speed I/O, and turtle graphics in limited memory.

Bound Volume 3: 1978

Item #015

The roots of the Silicon Valley growth. In 1978 Steve Wozniak and other programmers were publishing in *Dr. Dobb's Journal* code that would help them grow multi-million-dollar computer companies. The proposed S-100 bus standard was hashed out in *Dr. Dobb's* pages. *Dr. Dobb's* contributors began to speak more in terms of technique than of specific implementations as the industry began to diversify. Languages covered in depth included SAM76, Pilot, Pascal, and Lisp.

To Order:

To order any of Dr. Dobb's products, return the order form at the end of this catalog, or

CALL TOLL FREE 1-800-528-6050 EXT. 4001

and refer to product item number, title, and disk format.

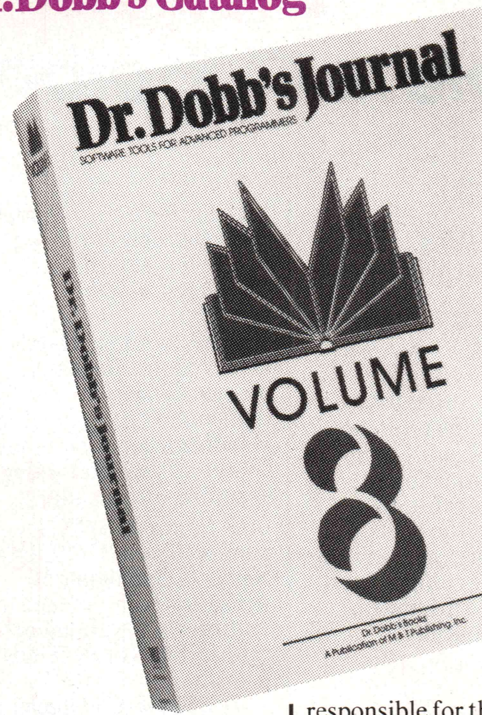
For customer service questions,

CALL M&T PUBLISHING, INC. 415-366-3600 EXT. 216

M&T BOOKS

Dr. Dobb's Catalog

Dr. Dobb's Bound Volumes



Bound Volume 4: 1979

Item #016

In the midst of the Gold Rush. Three years before IBM would release its PC, a thriving, rough-and-tumble personal computer industry existed. Fortunes had been made and lost, the effective power of the machine multiplied a hundredfold. By 1979 some stability had even emerged; one could speak of the processors that had proven longevity as microcomputer CPUs; the 8080, the Z80, the 6800, and the 6502. *Dr. Dobb's Journal* focused on the best ways to use these processors, with algorithms, tips, and code for 8- to 16-bit conversion, pseudo-random number generation, micro-to-mainframe connections, telecommunications, and networking. And lots of useful code.

Bound Volume 5: 1980

Item #017

The preeminence of CP/M and the rise of C. More than any other magazine, *Dr. Dobb's Journal* was responsible for the spread of CP/M and C on microcomputers. Both of those movements began in 1980. *Dr. Dobb's* all-CP/M issue, including Gary Kildall's history of CP/M, sold out within weeks of publication. This was the year of Ron Cain's original Small C complier, of a CP/M-oriented C interpreter, CP/M-to-UCSD Pascal file conversion techniques, ans a greater concern in *Dr. Dobb's* with software portability.

Bound Volume 6: 1981

Item #018

The first of Forth. 1981 saw *Dr. Dobb's* first all-Forth issue (now sold out), along with an emphasis on CP/M, C, telecommunications, and new languages, David Cortesi began "Dr. Dobb's Clinic," one of the magazine's most popular features. Highlights included information on PCNET, the Conference Tree, the Electronic Phone Book, Tiny Basic for the 6809, writing your own compiler, and a systems programming language.

Bound Volume 7: 1982

Item #019

Legitimacy. By 1982 IBM had become a player in the personal computer game and was changing the rules. New microprocessors arrived, the first designed speicifcally to serve as personal computer CPUs. In *Dr. Dobb's Journal* Dave Cortesi published the first serious comparison of MS DOS and CP/M-86. *Dr. Dobb's* started two new columns: the CP/M Exchange, as a rearguard maneuver to ensure that good tools for CP/M programmers would continue to be developed and circulated, and the 16-Bit Software Toolbox to investigate the 8088/86 and other new microprocessors. We published code for the 68000 and Z8000 processors, and looked ahead, in a provocative essay, to fifth-generation computers.

Bound Volume 8: 1983

Item #020

Power Tools. Personal computers were proving themselves to be true professional software development tools by 1983, the year in which Jim Hendrix completed his "canonical" version of Small C in *Dr. Dobb's Journal. Dr. Dobb's* published more 68000 and 8088 code, and as the memory limitations relaxed, the magazine's commitment to tight code let it shoehorn impossibly large systems into memory. Small C was just one of the major software products published in their entirety in *Dr. Dobb's* pages that year; there were Ed Ream's RED screen editor and a version of the Ada language called Augusta.

Buy the complete set and save 15%

If you buy all nine volumes, covering the entire editorial content of ***Dr. Dobb's Journal of Software Tools*** from the first issue in 1976 through 1984, you pay just $225. That's a 15% discount and over $40 off the combined individual prices. To order the complete set of Bound Volumes 1 through 9, ask for item #020C.

Vol. 1	Item #013	$27.75
Vol. 2	Item #014	$27.75
Vol. 3	Item #015	$27.75
Vol. 4	Item #016	$27.75
Vol. 5	Item #017	$27.75
Vol. 6	Item #018	$27.75
Vol. 7	Item #019	$30.75
Vol. 8	Item #020	$31.75
Vol. 9	Item #020B	$35.75
All 9 volumes Item #020C		$225.00

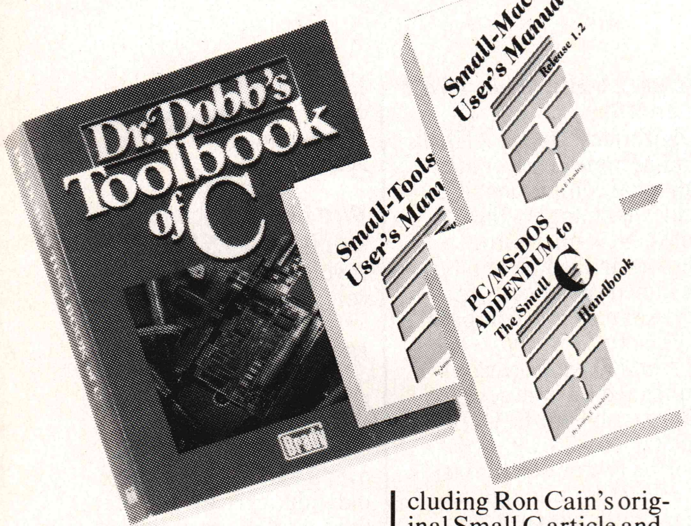
To Order:

To order any of Dr. Dobb's products, return the order form at the end of this catalog, or ***CALL TOLL FREE 1-800-528-6050 EXT. 4001*** *and refer to product item number, title, and disk format.*
For customer service questions, CALL M&T PUBLISHING, INC. 415-366-3600 EXT. 216

M&T BOOKS

A collection of powerful tools for C software developers, including books, software on disk, and reference materials from the publisher of Dr. Dobb's Journal Of Software Tools

Dr. Dobb's Complete C Toolbox



From M&T Publishing and Brady Communications . . .

Dr. Dobb's Toolbook of C

Item #005

The **Toolbook** contains over 700 pages of C material, including articles by such C experts as Kernighan and Ritchie, Cain and Hendrix, Skjellum and Holub. The level is sophisticated and pragmatic, appropriate for professional C programmers.

The most valuable part of the **Toolbook** to many will be the hundreds of pages of useful C source code, including a complete compiler, an assembler, and text-processing utilities. The accompanying text explains, in the programmers' own words, why they did what they did.

Dr. Dobb's Journal of Software Tools introduced a generation of personal computer programmers to the C programming language, and all the best C articles and code published in *Dr. Dobb's* over the years is included and updated in the **Toolbook**, including Ron Cain's original Small C article and articles from sold-out issues. But the **Toolbook** also includes material never before published, including Jim Hendrix's complete macro assembler in C.

Dr Dobb's offers the **Toolbook** in a **special hardbound edition** for just $29.95.

You'll find:
Jim Hendrix's famous **Small C Compiler** and **New Library for Small C** (both also available on disk), NEW: Hendrix's **Small Mac: An Assembler for Small C** and **Small Tools: Programs for Text Processing** (both also available on disk), All of Tony Skjellum's **C Programmer's Notebook** columns distilled by Tony into one thought-provoking chapter.

Dr. Dobb's C Software Tools on Disk

To complement the **Toolbook,** *Dr. Dobb's* also offers the following programs on disk. Full C source code and documentation is included. Except where indicated, both CP/M and MS/PC DOS versions are available.

Also from M&T Publishing and Brady Communications . . .

The Small C Handbook

Item #006 or #006A

Jim Hendrix's **Small-C Handbook** is the reference book on his Small-C compiler. In addition to describing the operation of the compiler, the book contains complete source listings to the compiler and its library of arithmetic and logical routines.

Hendrix has ported the compiler to the MS/PC DOS environment since the **Handbook** was printed, and the **Handbook** plus his **MS/PC DOS Handbook Addendum,** Item #006A, is $22.95.

A perfect companion to the Hendrix **Small-C compiler** offered by *Dr. Dobb's* on disk, the **Handbook** even tells how to use the compiler to generate a new version of itself.

While both the **Handbook** and the **Toolbook** provide documentation for the Small-C compiler, the **Handbook** contains a more detailed discussion and is available with an addendum for the MS/PC DOS version.

The **Handbook,** Item #006, is just $17.95. Jim

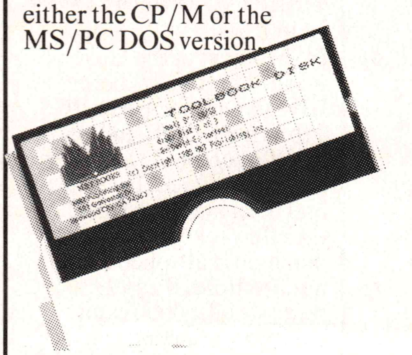
Small-C Compiler

Item #007

Jim Hendrix's Small-C Compiler is the most popular piece of software ever published in *Dr. dobb's* 10-year history. Like a home-study course in compiler design, the **Small-C Compiler** and **Small-C Handbook** provide everything you need but the computer for learning how compilers are constructed, and for learning C at its most fundamental level.

While both the **Handbook** and the **Toolbook** provide documentation for the Small-C compiler, the **Handbook** contains a more detailed discussion and is available with an addendum for the MS/PC DOS version. The **MC/PC DOS Small-C Handbook Addendum** is recommened in addition to the **Handbook** for MS DOS or PC DOS users.

The **Small-C compiler** is available for $19.95 in either the CP/M or the MS/PC DOS version.



To Order:

To order any of Dr. Dobb's products, return the order form at the end of this catalog, or

CALL TOLL FREE 1-800-528-6050 EXT. 4001

and refer to product item number, title, and disk format.

For customer service questions,

CALL M&T PUBLISHING, INC. 415-366-3600 EXT. 216

M&T BOOKS

Dr. Dobb's Complete C Toolbox

Small-Mac: An Assembler for Small-C
Item #012A

Small-Mac is an assembler designed to stress simplicity, portability, adaptability, and educational value. The package features a simplified macro facility, C language expression operators, object file visibility, descriptive error messages, and an externally-defined machine instruction table. You get the macro assembler, linkage editor, load-and-go loader, library manager, CPU configuration utility, and a utility to dump relocatable files.

Small-Mac is available with documentation for $29.95. For CP/M systems only.

Small-Tools: Programs for Text Processing
Item #010A

A package of programs performing specific, modular operations on text files, including: editing; formatting; sorting; merging; listing; printing; searching; changing; transliterating; copying; concatenating; encrypting and decrypting; replacing spaces with tabs and tabs with spaces; counting characters, words, or lines; and selecting printer fonts. Supplied in source code form so you can select and adapt these tools to your own purposes.

Small-Tools is available with documentation for $29.95. For CP/M or MS/PC DOS systems.

Special Packages—20% Off

Now for almost 20% off the combined individual product prices, you can order a complete set of *Dr. Dobb's* C programming tools for your CP/M or MS/PC DOS system.

C Package for CP/M
Item #005A

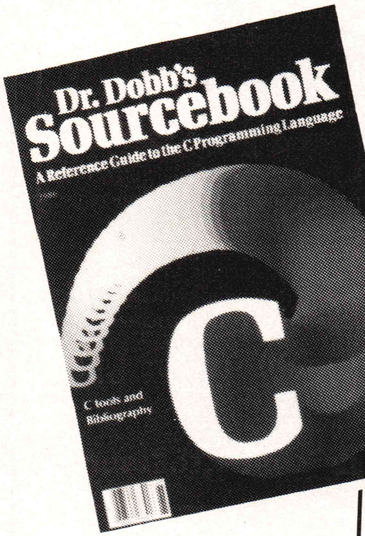
Ordered individually, these items would cost about $120. If you order the CP/MC package, you'll get ***Dr. Dobb's Toolbook,*** the **Small-C Handbook,** the **Small-C Compiler** on disk, the **Small-Mac** assembler on disk with documentation in the **Small-Mac Manual,** the **Small-Tools** text-processing programs on disk with documentation in the **Small-Tools Manual,** all for just $99.95.

C Package for MS/PC DOS
Item #005B

These items would cost over $100 if purchased individually. If you order the MS/PC DOS C package, you'll get ***Dr. Dobb's Toolbook,*** the **Small-C Handbook** with the **MS/PC DOS Addendum,** the **Small-C Compiler** on disk, the **Small-Tools** text-processing programs on disk with documentation in the **Small-Tools Manual,** all for just $82.95.

Dr. Dobb's Sourcebook: A Reference Guide to the C Programming Language
Item #004

Products and services for C programmers are appearing at so rapid a rate that it's all but impossible to keep up with them. *Dr. Dobb's* presents this handy



guide to the who, what, when, where, and why of C. A comprehensive reference manual to new information, products, and services, the **Sourcebook** contains:

- A bibliography of over 300 articles and books on the C language;
- A descriptive list of products for C programmers: compilers, editors, interpreters, and utilities;
- A list of C-related services: classes, seminars, and on-line services.

The **Sourcebook** is just $7.95. Dr. Dobb's Sourcebook: A Reference Guide to the C Programming Language
Item #004 $ 7.95

Dr. Dobb's Toolbook for C
Item #005 $29.95

The Small-C Handbook
Item #006 $17.95

The Small-C Handbook and MS/PC-DOS Addendum
Item #006A $22.95

Small-C Compiler disk
Item #007 $19.95

MS/PC DOS Small-C Handbook Addendum
Item # 008 $4.95

Small Tools: Programs for Text Processing disk & manual
Item #010A $29.95

Small Mac: An Assembler for Small-C disk (For CP/M only) & manual.
Item #012A $29.95

CP/M C Package
Item #005A $99.95

MS/PC DOS C Package
Item #005B $82.95

For CP/M disks, please specify one of the following formats: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8" SS/SD.

To Order:

To order any of Dr. Dobb's products, return the order form at the end of this catalog, or

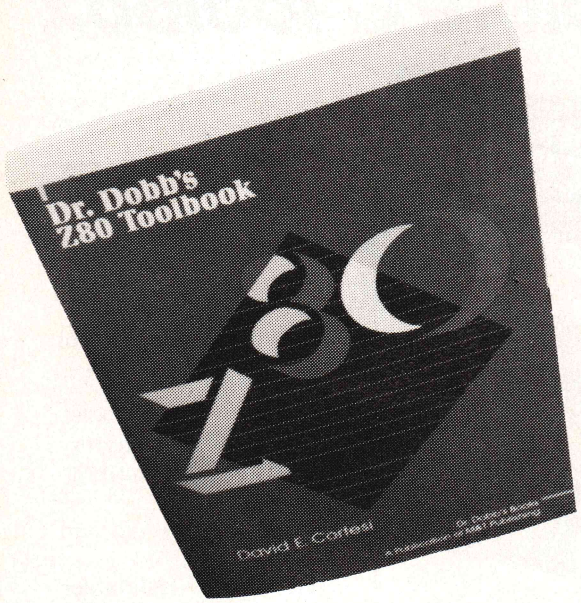
CALL TOLL FREE 1-800-528-6050 EXT. 4001

and refer to product item number, title, and disk format.
For customer service questions,

CALL M&T PUBLISHING, INC. 415-366-3600 EXT. 216

M&T BOOKS

Dr. Dobb's Z80 Toolbook



David E. Cortesi longtime Dr. Dobb's columnist and author of *Inside CP/M* brings you —

Dr. Dobb's Z80 Toolbook

Here's all you need to write your own Z80 assembly language programs for only $25!

Do you use CP/M? Do you feel as if the only part of the computer industry that has **not** abandoned you is your own Z80 computer? It keeps on working, but when you need programs for it, you have to write them yourself. When you do, you quickly find that while Pascal or BASIC is okay for some things, there is often no substitute for the speed, small size, and flexibility of an assembly language program.

Dr. Dobb's Z80 Toolbook puts the power of assembly language in the hands of anyone who's done a little programming. You'll find:

****A method of designing programs and coding them in assembly language.** Cortesi will take you on a walk through the initial specifications, designing an algorithm and writing the code. He demonstrates this method in the construction of several complete, useful programs.

****A complete, integrated toolkit of subroutines** for arithmetic, for string-handling, an for total control of the CP/M file system. They bring the ease and power of a compiler's runtime library to your assembly language work, without a compiler's size and sluggish code. **Best of all, every line of the toolkit's source code is there for you to read,** and every module's operation is explained with the clarity and good humor for which Dave Cortesi's writing is known.

Order the Z80 Software on Disk! Save Yourself the Frusrration of File Entry

All the software in **Dr. Dobb's Z80 Toolbook**—the programs plus the entire toolkit, both as source code and as object modules for both CP/M 2.2 and CP/M Plus—is yours on disk. (A Z80 microprocessor and a Digital Research International RMAC assembler or equivalent are required.)

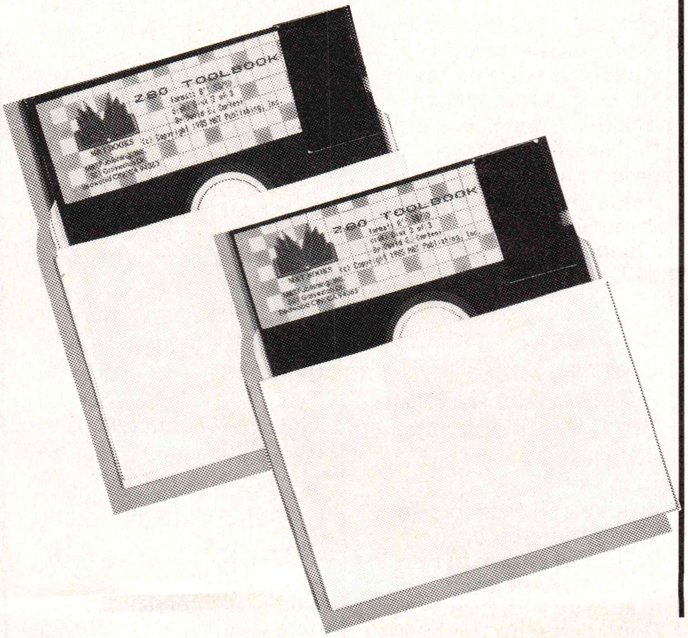
Receive *Dr. Dobb's Toolbook for Z80*, along with the software on disk, together for only $40!

Dr. Dobb's Toolbook for Z80
Item #022 $25

Dr. Dobb's Toolbook for Z80. together with software on disk. Please specify one of the following formats: 8" SS/SD; Apple; Osborne: Kaypro.

Item #022A $40

Most of the programs are included in the book; however, the disk is necessary for complete listings



To Order:

To order any of Dr. Dobb's products, return the order form at the end of this catalog, or

CALL TOLL FREE 1-800-528-6050 EXT. 4001

and refer to product item number, title, and disk format.
For customer service questions,

CALL M&T PUBLISHING, INC. 415-366-3600 EXT. 216

M&T BOOKS

Dr.Dobb's Catalog

A UNIX-like Shell and Utility Package for MS-DOS

A UNIX-like Shell for MS-DOS, V.2 and A UNIX-like Utility Package for MS-DOS by Dr. Dobb's C-Chest Columnist, Allen Holub

Only $29.95 each! If you are a registered user or have already purchased version 1 of The Shell from DDJ, you can receive the upgrade for only $6!

THE SHELL Version 2.0

An MS-DOS implementation of the most often used parts of the UNIX C Shell. This package includes an executable version of the shell, along with the **complete C source code and full documentation.** Supported features are:

Editing Command-line editing with the cursors is supported. The line is visible as you edit it.

Aliases Can be used to change the names of commands or as very fast, memory-resident, batch files. Nested aliases are supported in version 2.

History You can execute previous commands. The command can be edited before being executed. Version 2 supports embedded history requests (bar; !!>foo).

Redirection and Pipes
< > >>
>& >>&
|

Pipe temporary files can be put on a RAM disk.

Unix-like Command Syntax: / can be used to separate directory names (\ can now be used as well).

A 2048-byte command line is supported. Command-line wild card expansion. Multiple commands on a line.

DOS-compatible prompt support $d $t $ – $e $h $n $q $$ $%

C-Shell Based Shell Scripts (batch files) Shell Variables are macros that can be used on the command line. Version 2 supports arithmetic manipulation of shell variables using the @ command. The following C operators are supported:
( ) + - ★ / %
<= >= < > !=
== ! && || =

A batch file can call another batch file like a subroutine. Control is passed to the second file and then back to the first when the second is finished. Batch files can return values to the calling file using the exit and $status mechanisms.

A powerful, interpretive, programming language, based on the UNIX C Shell, is now supported, including:
if/then/else
while
foreach
switch/case
break
continue

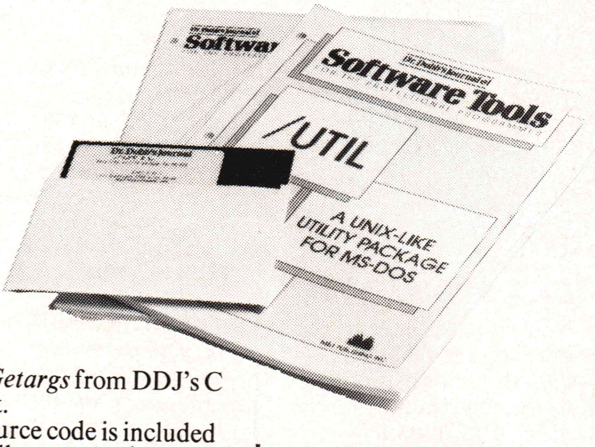
All commands can be nested.

/util

A UNIX-like Utility Package for MS-DOS This collection of utility programs for MS-DOS includes updates of the highly acclaimed Dr. Dobb's articles *Grep: A UNIX-Like Generalized Regular Expression Processor Ls* from *Dr. Dobb's* C Chest column, and *Getargs* from DDJ's C Chest.

Source code is included and all programs (and most of the utility subroutines) are fully documented in a UNIX-style manual. You'll find executable versions of:

cat A file concatenation and viewing program
cp A file copy utility
date Prints the current time and date
du Prints amount of space available and used on disk
echo Echoes its arguments to standard output
grep Searches for a pattern defined by a regular expression
Ls Gets a sorted directory
mkdir Creates a directory
mv Renames a file or directory. Moves files to another directory
p Prints a file, one page at a time
pause Prints a message and waits for a response
printenv Prints all the environment variables
rm Deletes one or more files
rmdir Deletes one or more directories
sub Text substitution utility. Replaces all matches of a regular expression with another string.
chmod change all file attributes (write permission, hidden, system, archive bit).



ORDER THE SHELL AND /UTIL TOGETHER FOR ONLY $50! SAVE OVER 15%

The Shell runs on IBM PC's and compatibles

The Shell Item #160	$29.95
Shell Upgrade Disk for owner of the Shell v.1 Item #160A	$ 6.00
/util Item #161	$29.95
Shell/util Package Item #162	$50.00

To Order:

To order any of Dr. Dobb's products, return the order form at the end of this catalog, or

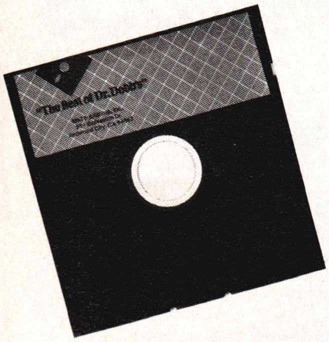
CALL TOLL FREE 1-800-528-6050 EXT. 4001

and refer to product item number, title, and disk format.
For customer service questions,

CALL M&T PUBLISHING, INC. 415-366-3600 EXT. 216

M&T BOOKS

Dr. Dobb's Listings On Disk and Back Issues



Dr. Dobb's Listings

You can save time entering Dr. Dobb's valuable code listings! Selected listings from *this August 1986 issue,* along with listings from all previous 1986 DDJ issues, are yours on disk!

Now, as a useful adjunct to the magazine, DDJ offers the additional value and convenience of selected listings from each 1986 issue, on disk. The first two of three Dr. Dobb's 1986 Listings disks, featuring a collection of listings from the DDJ January 1986 issue through *this August 1986 issue,* are now available.

And, in case you missed a single valuable issue of Dr. Dobb's Journal of Software Tools, while the supply lasts you can also receive any available 1986 back issue for only $2.50 each when you purchase a DDJ Listings disk. That's a savings of 50% off the regular back issues price!

Dr. Dobb's Listings Disk #1/86 January–April 1986 You'll find listings from the following articles, among others:

From issue #111 January 1986
***A Simple OS for Real-time Applications; 68000 assembly language techniques for an operating system kernel* by DDJ editor Nick Turner
***Exec calls and Fortran; a technique allowing execution of a user or system task from a user program* from DDJ's 16-Bit Software Toolbox, by Robert Sypek

***32-bit Square Roots; An 8086 assembly-language routine for 32-bit square roots* by Michael Barr
(Sorry, back issue #111 is sold out)

From Issue #112 February 1986
***Fast Integer Powers for Pascal; An implementation of the fastest-known algorithm for the computation of integer powers* by Dennis E. Hamilton
***Data Absrtaction with Modula-2; Construction of a priority queue in Modula-2* by Bill Walker and Stephen Alexander
***Learning Ada on a Micro; A draw poker program in Ada* by Do-While Jones
***Fast IBM PC graphics routines* from DDJ's 16-Bit Software Toolbox, by Dan Rollins
(Sorry, back issue #112 is sold out)

From Issue #113 March 1986
***Recursive Bose-Nelson Sort; An alternative to Joe Celko's September 1985 sort routine* by R.J. Wissbaum
***A Variable-Metric Minimizer; A C program for miniimizing arbitrary functions* by Joe Marasco
***Concurrancy and Turbo Pascal; An approach to implementing coroutine in Pascal* by Ernest Bergmann
***Speeding MS DOS Disk Access; Programs to test disk-access speed* by Greg Weissman
***Square Roots on the NS32000; Comparable square root routines in C and assembly language for National Semiconductor's 32000 family* by Richard Campbell

From Issue #114 April 1986
***Boca Raton Inference Engine; Lisp, Prolog, and Expert-2 techniques and code* by Robert Brown

Dr. Dobb's Listings Disk #2/86 May–August 1986
You'll find listings from the following articles, among others.

From Issue #115 May 1986
***Simple plots with the Enhanced Graphics Adapter* by Nabajyoti Barkakati
***The Cryptographer's Toolbox* by Fred A. Scacchitti

From Issue #116 June 1986
***Structured Programming: Overloading Procedures, Exporting Opaque Types, Data Hiding* by Namir Shammas
***Compuserve B Protocol* by Steve Wilhite

From Issue #117 July 1986
***Structured Programming: Tiny Tools, Array-Defining Words* by Michael Ham

From Issue #118 August 1986
***Structured Programming: Generic Routines in Ada and Modula-2, Extended for Loop* by Namir Shammas

Dr. Dobb's Listings Disk #1/86
Item #170 $14.95
Dr. Dobb's Listings Disk #2/86
Item #171 $14.95

Please specify MS/DOS, Macintosh, or CP/M. For CP/M disks, please specify one of the following formats: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8″ SS/SD.

To order any DDJ 1986 back issue at the discounted price of only $2.50 each with your Dr. Dobb's Listings Disk, please specify the issue date and number on the order form. January Issue #111 and February Issue #112 are sold out.

To Order:

To order any of Dr. Dobb's products, return the order form at the end of this catalog, or

CALL TOLL FREE 1-800-528-6050 EXT. 4001

and refer to product item number, title, and disk format.
For customer service questions,

CALL M&T PUBLISHING, INC. 415-366-3600 EXT. 216

M&T BOOKS

Dr.Dobb's Catalog

Order Form

ORDER NOW

NAME ______________________________

(Please use street address, not P.O. Box)

ADDRESS ______________________________

CITY ______________ STATE ______ ZIP ______

DAY PHONE ______________________________

For Faster Service
On Credit Card Orders

CALL TOLL FREE
1-800-528-6050
Ext. 4001

Please Refer To Item #
When Ordering

Or, Fill Out This Postage Paid,
Self-Mailing Order Form
And Return To:

M&T PUBLISHING INC.
501 GALVESTON DRIVE
REDWOOD CITY, CA 94063

For disk orders, please indicate format. Refer to ad for standard format availability for each product.
Special formats available for additional $10 each.

☐ MS/DOS ☐ Macintosh ☐ Apple II ☐ CP/M
____ Zenith Z-100 DS/DD
____ Apple
____ Osborne
____ Kaypro
____ 8" SS/SD

QUANTITY	ITEM #	DESCRIPTION	UNIT PRICE	TOTAL PRICE

SUB-TOTAL	
SALES TAX	
SHIPPING	
TOTAL ORDER	

CA residents must add applicable sale tax on merchandise total ________ %
(CA residents must add sales tax EXCEPT *Dr. Dobb's Sourcebook* #004)
Shipping must be included with order. See rates below.

In U.S. For Bound Volumes, add $2.25 per book. Add $8.75 for special C Packages. For other books and disks, add $1.75 per item.

Outside U.S. For Bound Volumes, add $5.25 per book surface mail. Add $18 surface mail for Special C Packages. For other books and disks, add $3.25 per item surface mail. Foreign airmail rates available on request.

☐ VISA

☐ MASTERCARD

☐ AMERICAN EXPRESS

☐ CHECK

(make checks payable to M&T Publishing)

NAME ON CARD ______________________________

ACCOUNT NO. ______________________________

EXPIRATION DATE ______________________________

SIGNATURE ______________________________

PROMPT DELIVERY! DEALER INQUIRY WELCOME!

M&T BOOKS

3119S

Dr. Dobb's Catalog Order

Please Rush!

Please fold along fold-line and staple or tape closed.

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

First Class Permit No. 790 Redwood City, CA

Postage Will Be Paid By Addressee

Dr. Dobb's Catalog
501 Galveston Dr.
Redwood City, CA 94063

Please fold along fold-line and staple or tape closed.

Listing One (Listing continued, text begins on page 24.)

```
      {
        case 'A':        /* "-a" option */
          aflag = TRUE;
          if(argc > 1 && is_float(*(argv+1)))
          {
          argc--;
          argv++;
          if((a_val = atof(*argv)) <= 0.00)
            error(ILL_ARG,*argv);
        }
        break;
      case 'K':        /* "-k" option */
        if(pflag == TRUE)
          error(ILL_CMB,NULL);
        kflag = TRUE;
        if(argc > 1 && is_float(*(argv+1)))
        {
          argc--;
          argv++;
          k_val = atof(*argv);
          if(k_val < 0.00)
            error(ILL_KVL,*argv);
          break;
        }
        else
          error(MIS_KVL,NULL);
      case 'N':        /* "-n" option */
        if(argc > 1)
        {
          argc--;
          argv++;
          if((n_val = atoi(*argv)) < 1)
            error(ILL_NVL,*argv);
          else
            break;
        }
        else
          error(MIS_NVL,NULL);
      case 'P':        /* "-p" option */
        if(kflag == TRUE)
          error(ILL_CMB,NULL);
        pflag = TRUE;
        break;
      case 'X':        /* "-x" option */
        x1flag = TRUE;
        if(argc > 1 && is_float(*(argv+1)))
        {
          argc--;
          argv++;
          x1_val = atof(*argv);
        }
        else
          error(MIS_XVL,NULL);
        if(argc > 1 && is_float(*(argv+1)))
        {
          x2flag = TRUE;
          argc--;
          argv++;
          x2_val = atof(*argv);
          if(x2_val <= x1_val)
            error(ILL_XVL,x2_val);
        }
        break;
      default:         /* "-n" option */
        error(ILL_OPT,*argv);
    }
}
if(n_val == 0)          /* Set "n_val" if not given */
  n_val = 100;
/* Get the input data */

while(1)                /* ... while there is more input data */
{
  if(aflag == TRUE)     /* Automatic abscissae were called for */
  {
    if(n == 0)
      x[0] = x1_val;
    else
      x[n] = x[n-1] + a_val;
  }
  else                  /* Abscissae supplied with input data */
  {
    if(gets(buffer))
      x[n] = atof(buffer);
    else
      break;
  }
  if(gets(buffer))      /* Read in the corresponding ordinate */
    y[n] = atof(buffer);
  else
  {
    if(aflag == TRUE)
      break;
    else
      error(MIS_YVL,NULL);
  }
  if(++n == MAX_SIZE) /* Maximum amount of input data? */
    break;
}
if(n < 4)              /* Check for insufficient input data */
  error(INS_INP,NULL);

/* Check for non-monotonic abscissae. Output input data set
 * without interpolation if true.
 */

h = x[1] - x[0];
for(i = 1; i < n-1; i++)
  if(fabs(x[i+1] - x[i] - h) > 0.0)
  {
    for(i = 0; i < n; i++)
      printf("%g\n%g\n",x[i],y[i]);
    exit();
  }

/* Calculate abscissa interval. Use "-x" option values if
 * they were given unless they fall outside the range of
 * given (or calculated) abscissae.
 */

if(x1flag == FALSE || x1_val < x[0])
  x1_val = x[0];
if(x2flag == FALSE || x2_val > x[n-1])
  x2_val = x[n-1];
x_intvl = (x2_val - x1_val)/n_val;

/* Find the coefficients */

if(pflag == FALSE)
  spl_coeff(y,d2y,h,n,k_val);
else
  pspl_coeff(y,d2y,h,n);

/* Interpolate and output results */

ix = x1_val;
i = 1;
for(j = 0; j <= n_val; j++)
{
  while(ix >= x[i] && i < n - 1)
    i++;
  iy = spl_int(ix,x,y,d2y,h,i);
  printf("%g\n%g\n",ix,iy);
  ix += x_intvl;
}
}

/*** Functions ***/

/* SPL_COEFF() - Calculate spline coefficients and return in
 *               vector "d2y[]". Matrix to be solved has the
 *               typical form:
 *
 *      +-                  -+ +-   -+        +-                -+
 *      | 4+k  1    0    0   | | y1" | = m *  | y2 - 2*y1 + y0   |
 *      | 1    4    1    0   | | y2" |        | y3 - 2*y2 + y1   |
 *      | 0    1    4    0   | | y3" |        | y4 - 2*y3 + y2   |
 *      | 0    0    1   4+k  | | y4" |        | y5 - 2*y4 + y3   |
 *      +-                  -+ +-   -+        +-                -+
 *
 *
 *               where k = k_val, m = 6.0/(h*h) and yn" is the
 *               second derivative of the interpolated function
 *               at the "nth" abscissa, y0" = k * y1" and
 *               y5" = k * y4".
 */

void spl_coeff(y,d2y,h,n,k_val)
float y[];
double d2y[],
       h,
       k_val;
int n;
{
  double m,
         a[MAX_SIZE-1];
  int i;

  /* Set up the (symmetric tridiagonal) matrix, where the only
```

(continued on next page)

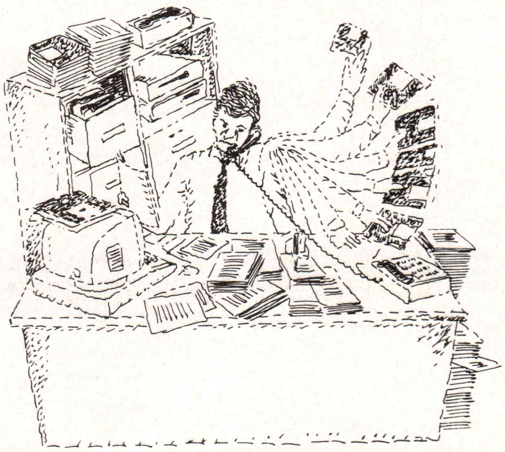
Listing One (Listing continued, text begins on page 24.)

```
    * elements of interest are those on the diagonal. These are
    * stored in array "a[]". Array "d2y[]" initially holds the
    * constants vector, then is overlaid with the calculated
    * variables vector.
    */

  m = 6.0/(h*h);
  for(i = 1; i < n-1; i++)
    d2y[i] = m * (y[i+1] - 2.0 * y[i] + y[i-1]);
  a[1] = 4.0 + k_val;

  /* Reduce the matrix to upper triangular form */

  for(i = 2; i < n-2; i++)
  {
    a[i] = 4.0 - 1.0/a[i-1];
    d2y[i] -= d2y[i-1]/a[i-1];
  }
  a[n-2] = 4.0 + k_val - 1.0/a[n-3];
  d2y[n-2] -= d2y[n-3]/a[n-3];
  d2y[n-2] /= a[n-2];

  /* Solve using back substitution */

  for(i = n-3; i > 0; i--)
    d2y[i] = (d2y[i] - d2y[i+1])/a[i];

  /* Solve for end conditions */

  d2y[n-1] = d2y[n-2] * k_val;
  d2y[0] = d2y[1] * k_val;
}

/* PSPL_COEFF() - Calculate periodic spline coefficients and
 *                return in vector "d2y[]". Matrix to be solved
 *                has the typical form:
 *
 *        +-                -+ +-   -+      +-                   -+
 *        | 4  1  0  0  1 | | y1" | = m * | y2 - 2*y1 + y0      |
 *        | 1  4  1  0  0 | | y2" |       | y3 - 2*y2 + y1      |
 *        | 0  1  4  1  0 | | y3" |       | y4 - 2*y3 + y2      |
 *        | 0  0  1  4  1 | | y4" |       | y5 - 2*y4 + y3      |
 *        | 1  0  0  1  4 | | y5" |       | y1 - y0 - y5 + y4   |
 *        +-                -+ +-   -+      +-                   -+
 *
 *              where m = 6.0/(h*h) and yn" is the second
 *              derivative of the interpolated function at the
 *              "nth" abscissa and y0" = y5".
 */

void pspl_coeff(y,d2y,h,n)
float y[];
double d2y[],
       h;
int n;
{
  double c,
         m,
         fabs();
  static double a[MAX_SIZE-1],
                b[MAX_SIZE];
  int i;

  /* Check for matching end ordinates */

  if(fabs(y[n-1] - y[0]) > 0.0)
    error(NMT_ORD,NULL);

  /* Set up the matrix, where array "a[]" holds the diagonal
   * elements, "b[]" holds those elements of column "n-1", and
   * "c" holds the current element of interest of row "n-1".
   * Array "d2y[]" initially holds the constants vector, then is
   * overlaid with the calculated variables vector.
   */

  m = 6.0/(h*h);
  for(i = 1; i < n-1; i++)
    d2y[i] = m * (y[i+1] - 2.0 * y[i] + y[i-1]);
  d2y[n-1] = m * (y[1] - y[0] - y[n-1] + y[n-2]);
  a[1] = 4.0;
  b[1] = 1.0;
  b[n-2] = 1.0;
  b[n-1] = 4.0;
  c = 1.0;

  /* Reduce the matrix to upper triangular form */

  for(i = 2; i < n-1; i++)
  {
    m = 1/a[i-1];
    a[i] = 4.0 - m;
    b[i] -= b[i-1] * m;
    d2y[i] -= d2y[i-1] * m;
    b[n-1] -= c * m * b[i-1];
    d2y[n-1] -= c * m * d2y[i-1];
    c = -c * m;
  }
  c += 1.0;
  b[n-1] -= c * b[n-2]/a[n-2];
  d2y[n-1] -= c * d2y[n-2]/a[n-2];

  /* Solve using back substitution */

  d2y[0] = d2y[n-1] /= b[n-1];
  d2y[n-2] = (d2y[n-2] - b[n-2] * d2y[n-1])/a[n-2];
  for(i = n-3; i > 0; i--)
    d2y[i] = (d2y[i] - b[i] * d2y[n-1] - d2y[i+1])/a[i];
}

/* SPL_INT - Interpolate points using spline function */

double spl_int(ix,x,y,d2y,h,i)
float x[],
      y[];
double ix,
       d2y[],
       h;
int i;
{
  double iy,
         t1,
         t2;

  t1 = (ix - x[i-1])/h;
  t2 = (x[i] - ix)/h;
  iy = y[i-1] * t2 + y[i] * t1 - SQUARE(h) * (d2y[i-1] * (t2 -
       CUBE(t2)) + d2y[i] * (t1 - CUBE(t1)))/6.0;
  return iy;
}

/* IS_FLOAT() - Check that character string is in correct floating
 *              point format. Return TRUE if correct, FALSE
 *              otherwise. The algorithm used is a deterministic
 *              finite state machine. Using the regular
 *              expression terminology of Unix's "lex", the
 *              character string must be of the form:
 *
 *                   -?d*.?d*(e|E(\+|-)?d+)?
 *
 *              where d = 0|1|2|3|4|5|6|7|8|9
 */

BOOL is_float(str)
char *str;
{
  int c,                /* Next FSM input character */
      state = 0;        /* Current FSM state */

  while(c = *str++)
  {
    switch(state)
    {
      case 0:           /* FSM State 0 */
        switch(c)
        {
          case '-':
            state = 1;
            break;
          case '.':
            state = 3;
            break;
          default:
            if(isdigit(c))
              state = 2;
            else
              return FALSE;
            break;
        }
        break;
      case 1:           /* FSM State 1 */
        switch(c)
        {
          case '.':
            state = 2;
            break;
          default:
            if(isdigit(c))
              state = 2;
```

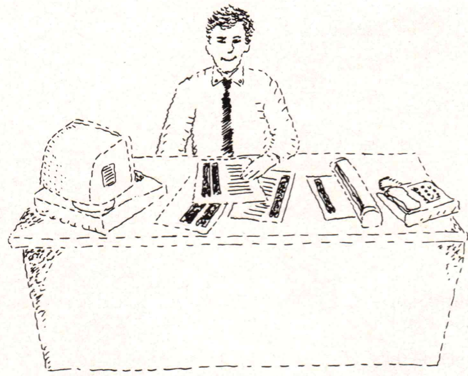
```
                else
                  return FALSE;
                break;
          }
          break;
        case 2:             /* FSM State 2 */
          switch(c)
          {
            case '.':
              state = 4;
              break;
            case 'e':
            case 'E':
              state = 5;
              break;
            default:
              if(isdigit(c))
                state = 2;
              else
                return FALSE;
              break;
          }
          break;
        case 3:             /* FSM State 3 */
          if(isdigit(c))
            state = 4;
          else
            return FALSE;
          break;
        case 4:             /* FSM State 4 */
          switch(c)
          {
            case 'e':
            case 'E':
              state = 5;
              break;
            default:
              if(isdigit(c))
                state = 4;
              else
                return FALSE;
              break;
          }
          break;
        case 5:             /* FSM State 5 */
          switch(c)
          {
            case '+':
            case '-':
              state = 6;
              break;
            default:
              if(isdigit(c))
                state = 7;
              else
                return FALSE;
              break;
          }
          break;
        case 6:             /* FSM State 6 */
          if(isdigit(c))
            state = 7;
          else
            return FALSE;
          break;
        case 7:             /* FSM State 7 */
          if(isdigit(c))
            state = 7;
          else
            return FALSE;
          break;
      }
  }
  return TRUE;
}

/* ERROR() - Error reporting. Returns an exit status of 2 to the
 *          parent process.
 */

void error(n,str)
int n;
char *str;
{
  fprintf(stderr,"\007\n*** ERROR - ");
  switch(n)
  {
    case ILL_ARG:
      fprintf(stderr,"Argument must be greater than zero: %s",
          str);
      break;
    case ILL_CMB:
      fprintf(stderr,"Cannot use -k option with -p option");
      break;
    case ILL_KVL:
      fprintf(stderr,"Illegal value for -k option: %s",str);
      break;
    case ILL_NVL:
      fprintf(stderr,"Illegal value for -n option: %s",str);
      break;
    case ILL_OPT:
      fprintf(stderr,"Illegal command line option: %s",str);
      break;
    case ILL_XVL:
      fprintf(stderr,"Illegal value for -x option: %s",str);
      break;
    case INS_INP:
      fprintf(stderr,"Insufficient input data");
      break;
    case MIS_KVL:
      fprintf(stderr,"Missing value for -k option");
      break;
    case MIS_NVL:
      fprintf(stderr,"Missing value for -n option");
      break;
    case MIS_XVL:
      fprintf(stderr,"Missing value for -x option");
      break;
    case MIS_YVL:
      fprintf(stderr,"Missing ordinate value");
      break;
    case NMT_ORD:
      fprintf(stderr,"End ordinates do not match");
      break;
    default:
      break;
  }
  fprintf(stderr," ***\n\nUsage: spline [-aknpx]\n");
  exit(2);
}

/*** End of SPLINE.C ***/
```

End Listing

SOFTSTRIP®
NOW OFFERS YOU SOMETHING IN NEVER BEFORE AVAILABLE...



CONVENTIONAL DATA HANDLING



THE SOFTSTRIP SYSTEM

A CHOICE.

Until now you were stuck with disks.

No more. Install our unique STRIPPER™ software on your personal computer today and discover the many benefits of the fastest, easiest, least expensive way to handle information.

STRIPPER lets you print – ON PAPER – your own machine readable Softstrip data strips using your dot matrix printer. The Softstrip System Reader reads that information into a computer rapidly. With STRIPPER and the reader, your PC and printer become part of the most versatile information handling system available.

With this system you can do anything you wish with any data you have in your PC – ON PAPER.

DATA ENTRY: Why use keystrokes when you can eliminate them with data strips? Whatever the document - invoices, packing slips, memos, letters, sales reports, the list is endless — simply print a data strip right on the same printed page. Now you have a document that is both human readable and machine readable. A typical document can be entered in only 15 seconds using data strips. And, it ends keystroke errors forever.

DATA DISTRIBUTION: Why copy disks? It's time consuming and expensive. Softstrip data strips will end all that. Simply photocopy as many data strips as you like and send them by mail. Data strips ignore folding, coffee stains, ink marks and, by the way, magnetic fields. And if you're using telecommunications, you can stop making the phone company rich.

DATA STORAGE AND RETRIEVAL: Why have a file of disks and a file of paper? Eliminate one with Softstrip data strips. File the data strip with the document. Better still, print the strip right on the document. Then put it in a file or binder.

Retrieval is simple. To find existing data, pull the document and its related data strip from the file. They've been stored together. Then use the reader to enter the data. No more hassle trying to match documents with the right disk — if you can find it.

DATA TRANSFER: Why bother with cables, modems and phone lines to move files between computers? A Softstrip data strip generated by an IBM PC can be read into another PC, or compatible, an Apple or even a Macintosh. If you work at home on a Macintosh, make a data strip on your printer, take it into the office and read it into your IBM PC. Simple. And we've created the utilities to let you do that easily. (See Application Notes on opposite page.)

Fascinating, isn't it? Anything you can do with disks can be done with the Softstrip data strip system — faster, easier and at lower cost — ON PAPER.

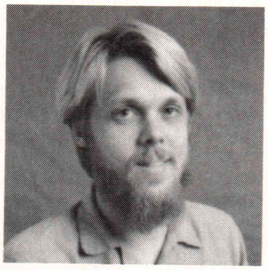
All you need is STRIPPER software at **$19.95** and the Softstrip System Reader at **$199.95.**

Softstrip®
COMPUTER READABLE PRINT

The PC User's Edge™

DATA HANDLING

SEPTEMBER CASE HISTORY



Nick Turner, editor of *Dr. Dobbs Journal,* has been making his own data strips to back up and store articles, listings and other important materials. Turner notes that the STRIPPER™ software "creates compact, machine readable archives that are immune to dust, dirt and magnetic fields". Strips that Turner prints on his printer are filed in a loose leaf binder, along with the appropriate article. This permits him to pull the strip from the binder and read it back into his computer quickly using the reader. The method saves disk file space, since once an article has been printed, maintaining it on a disk for revision isn't required.

The STRIPPER system has solved another problem for the busy editor, inter-machine file compatibility. Using the Softstrip® System, Turner can transfer files between his PC-clone and his Macintosh "with a minimum of fuss".

"It's about time we had a way to store files on paper in machine readable form," Turner comments. Mr. Turner's comments are his own and do not reflect the opinion of the publication.

APPLICATION NOTES

On the other side of this ad we said you can move data between different programs — on paper.

Using Softstrip data strips you can!

For instance move data between AppleWorks and Excel and back. Or Lotus 1-2-3 to and from AppleWorks.

We've created a series of several dozen Application Notes on Softstrip data strips. These lead you through simple steps to make the file transfer as easy as possible.

Here are just a few examples:

- AppleWriter to or from MacWrite.
- dBASE to or from Appleworks.
- WordStar to or from AppleWriter.
- WordStar to or from MacWrite.
- Framework to PageMaker.

And more are coming.

These strips contain "The IRA Calculator," a work-sheet for calculating returns with either Lotus or Excel. To receive the complete Application Note, call 1-800-533-7323, or write to Cauzin.

ACT NOW!! Don't delay.
See your local SOFTSTRIP dealer or call us at 1-800-533-7323.
In Connecticut: 203-573-0150.

Users' Groups: Call for special User Group deals.

CAUZIN
835 South Main Street
Waterbury, CT 06706
(203) 573-0150

Circle **no. 226** on reader service card.

1 2

Softstrip
COMPUTER READABLE PRINT

Listing One (Text begins on page 32.)

```
/* Listing one */
/*Radix sort listing*/

/* These includes are necessary to get the proper declarations for some of
   the Macintosh routines */
#include event.h
#include quickdraw.h

typedef struct {            /*Define a structure for the sort keys*/
        struct sortrec *ptr;        /*Pointer to next key*/
        char sortdat[KEYSIZ];       /*The key, containing KEYSIZ bytes*/
        } sortrec;
char *lmalloc();

main()

{       sortrec *begin,*first,*start,*temp;
        int i,j,recno;
        long tick;
        unsigned char dat;
        MaxApplZone();                      /*Get maximum size application heap*/
        begin = (sortrec *) lmalloc(0x40000); /*Allocate 256K for sortrecs*/
       /*Check if allocation was successful and exit if not*/
       if (!begin) {
                printf("\nNot enough memory in heap");
                _exit();
                }
        first = begin;                      /*Point to first record*/
        printf("How many recs to sort\n");
        scanf ("%d",&recno);
        for (i=0; i<recno; ++i) {
                first->ptr = first+1;       /*Set pointer to next record in heap*/
                for (j=0; j< KEYSIZ; ++j)    /*Fill with KEYSIZ random bytes*/
                        first->sortdat[j] = (Random()&0x7fff)%256;
                ++first;            /*Point to next sort key record*/
                }                           /*All records allocated and filled*/
        first->ptr = 0;                     /*Terminate the linked list*/
        tick = TickCount();                 /*Get the current time*/
        start = sort(KEYSIZ,begin); /*Sort the list*/
        printf("\nTickcount=%ld",TickCount()-tick); /*Print elapsed time*/
        }

 sortrec *sort(a,b)
 int a;             /*number of bytes in the sort key*/
 sortrec *b;        /*Pointer to head of linked list to be sorted*/

 {      /*First and last are pointers to follow two linked lists whose
          heads are start and start2.  Temp follows the full-length
          original or partly sorted list*/
 sortrec *first,*last,*start,*temp,*start2;
        static char mask[8] = { 1,2,4,8,16,32,64,128};/*Individual bit masks*/
        register i,j;

        start = b;                          /*point to original unsorted list*/
        for (i = a-1; i >= 0; --i) { /*Loop for all bytes of the sort key*/
                for (j = 0; j < 8; ++j) { /*Loop for all bits of each byte*/
                        first = last = start2 = 0;  /*Set up working ptrs*/
                        /*Loop for each key in the list*/
                        for (temp = start; temp; temp = temp->ptr) {
                                if (temp->sortdat[i]&mask[j])
        /*Value of the bit was 1.  If last list is empty, initialize it*/
                                        if (last==0) start2 = temp;
                                        /*else add this item to the list*/
                                        else last->ptr = temp;
                                        last = temp;
                                        }

        /*Value was 0. If first list empty, initialize it*/
                                else {
                                        if (first==0) start = temp;
                                        /*else add this item to the list*/
                                        else first->ptr = temp;
                                        first = temp;
                                        }
                                } /*End of list*/
                        /*If last list not empty, terminate it*/
                        if (last) last->ptr = 0;
                        /*if first list empty, use last only*/
                        if (first == 0)start = start2;
                        /*Else add last list to first list*/
                        else first->ptr = start2;
                        } /*All bits this byte examined*/
                        } /*all bytes examined*/
                return start;
                }
```

End Listing One

Listing Two

```
/* Listing two */
/*Shell sort listing*/

/*Includes for certain Macintosh routines*/
#include quickdraw.h
#include event.h
typedef struct {   /*This structure consists only of KEYSIZ bytes*/
        char sortdat[KEYSIZ];
        } sortrec;
char *lmalloc();

main()

{       int i,j,recno;
        long tick;
        sortrec *array, *base;
        unsigned char dat;
        MaxApplZone();     /*Get heap space, allocate 256K for sort keys*/
        array = base = (sortrec *) lmalloc(0x40000);
        printf("How many recs to sort\n");
        scanf ("%d",&recno);
        if (!array) {
                printf("\nNot enough memory in heap");
                exit();
                }
/*Fill the area with random data.  Use array as a pointer to all records*/
        for (i=0; i<recno; ++i) {
                for (j=0;j<KEYSIZ;++j) {
                        dat = (Random()&0x7fff)%256;
                        array->sortdat[j] = dat;
                        }
                ++array;
                }
        tick = TickCount();                     /*Get the current time*/
        sort(KEYSIZ,base,comp,swap,recno);
        printf("\nTickcount=%ld",TickCount()-tick);    /*Print elapsed time*/
        }

sortrec *sort(size,start,comp,swap,n)
int size,n;
sortrec *start;
long (*comp)(),(*swap)();

/*This is essentially identical to that in the  Kernighan and Ritchie text.
  The call includes size (the number of records), start (a pointer to the
  first record), comp and swap (routines for comparing and swapping byte
  strings, given pointers to them, and the number of bytes contained therein,
  and n (the number of bytes in the sort key)*/

{       int gap,i,j;
        for (gap = n/2; gap > 0; gap /= 2)
                for (i=gap; i<n; i++)
                        for (j=i-gap; j>=0; j-=gap) {
                                if ((*comp)(start+j,start+j+gap,size) <= 0) break;
                                (*swap)(start+j,start+j+gap,size);
                                }
        }

comp(val1,val2,n)
char *val1,*val2;
int n;

{       register i;
        for (i=0; i<n; ++i) {
                if (*(val1+i) < *(val2+i)) return (-1);
                else if (*(val1+i) > *(val2+i)) return(1);
                }
        return 0;
        }

swap(val1,val2,n)
char *val1,*val2;
int n;
{       register i,temp;
        for (i=0; i<n; ++i) {
                temp = *val1;
                *val1++ = *val2;
                *val2++ = temp;
                }
        }
```

End Listings

HIGH SPEED THRILLS

Listing One

(Text begins on page 46.)

```
These listing are available for downloading from co-author Mike
Elkins Fido BBS at (619)722-8724 which operates at 300 and 1200
baud, 8-bits, no partity, 1 stop-bit.

---- LISTING 1 ---------------------------------------------------------------

/*
   Erathosthenes Sieve Prime Number Program in C
*/
#define TRUE 1
#define FALSE 0
#define SIZE 8190
#define SIZEP1 8191
char flags[SIZEP1];
main()
 {
 int i, prime, j, count, loops;

 loops = 1;
 while(loops <= 10)    /* Changed to 100 for second benchmark */
  {
  count = 0;
  i = 0;
  while(i <= SIZE)
   {
   flags[i] = TRUE;
   i++;
   }
  i = 0;
  while(i <= SIZE)
   {
   if(flags[i])
    {
    prime = i + i + 3;
    j = i + prime;
    while(j <= SIZE)
     {
     flags[j] = FALSE;
     j = j + prime;
     }
    count++;
    }
   i++;
   }
  loops++;
  }
 printf("%d primes, %d loops\n", count, loops);
 return;
 }
```

End Listing One

Listing Two

```
----- LISTING 2 --------------------------------------------------------------
/*
 *      "DHRYSTONE" Benchmark Program
 *
 *      Compile:        cl dry.c  (Microsoft 3.0)
 *      Defines:        Defines are provided for old C compiler's
 *                      which don't have enums, and can't assign structures.
 *                      The time(2) function is library dependant; One is
 *                      provided for CI-C86.  Your compiler may be different.
 *                      This is not required for Microsoft 3.0 which supports
 *                      all of the standard calls and will compile asis.
 *
 *
 *      MACHINE                 OPERATING        COMPILER         DHRYSTONES
 *      TYPE                    SYSTEM                             /SEC
 *      ---------------         ------------     ---------------  -----------
 *      IBM PC                  PCDOS 3.1        Microsoft C 3.0    333
 *      IBM PC/AT               PCDOS 3.1        Microsoft C 3.0   1041
 *    $ IBM PC/AT               PCDOS 3.0        CI-C86 2.1         684
 *    $ ATT 3B2/300             UNIX 5.2         cc                 806
 *    $ IBM PC/AT               VENIX/86 2.1     cc                1000
 *    $ Sun2/120                Sun 4.2BSD       cc                1219
 *
 *    The entries with $ supplied by Rick Richardson, who origanally converted
 *    the program from ADA.
 *    The rest are provided by Mike's "C" Board 619-722-8724 .
 ****************************************************************************
 *
 *      The following program contains statements of a high-level programming
 *      language (C) in a distribution considered representative:
 *
 *      assignments                     53%
 *      control statements              32%
 *      procedure, function calls       15%
 *
 *      100 statements are dynamically executed.  The program is balanced with
 *      respect to the three aspects:
 *              - statement type
 *              - operand type (for simple data types)
 *              - operand access
```

```
 *                     operand global, local, parameter, or constant.
 *
 *      The combination of these three aspects is balanced only approximately.
 *
 *      The program does not compute anything meaningfull, but it is
 *      syntactically and semantically correct.
 *
 */

/* Compiler dependent options */
#undef  NOENUM                  /* Define if compiler has no enum's */
#undef  NOSTRUCTASSIGN          /* Define if compiler can't assign structures */
#undef  NOTIME                  /* Define if no time() function in library */

#ifdef  NOSTRUCTASSIGN
#define structassign(d, s)      memcpy(&(d), &(s), sizeof(d))
#else
#define structassign(d, s)      d = s
#endif

#ifdef  NOENUM
#define Ident1  1
#define Ident2  2
#define Ident3  3
#define Ident4  4
#define Ident5  5
typedef int     Enumeration;
#else
typedef enum    {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
#endif

typedef int     OneToThirty;
typedef int     OneToFifty;
typedef char    CapitalLetter;
typedef char    String30[31];
typedef int     Array1Dim[51];
typedef int     Array2Dim[51][51];

struct  Record
{
        struct Record           *PtrComp;
        Enumeration             Discr;
        Enumeration             EnumComp;
        OneToFifty              IntComp;
        String30                StringComp;
};

typedef struct Record   RecordType;
typedef RecordType *    RecordPtr;
typedef int             boolean;

#define NULL            0
#define TRUE            1
#define FALSE           0

#ifndef REG
#define REG
#endif

extern Enumeration      Func1();
extern boolean          Func2();

main()
{
        Proc0();
}

/*
 * Package 1
 */
int             IntGlob;
boolean         BoolGlob;
char            Char1Glob;
char            Char2Glob;
Array1Dim       Array1Glob;
Array2Dim       Array2Glob;
RecordPtr       PtrGlob;
RecordPtr       PtrGlobNext;

Proc0()
{
        OneToFifty              IntLoc1;
        REG OneToFifty          IntLoc2;
        OneToFifty              IntLoc3;
        REG char                CharLoc;
        REG char                CharIndex;
        REG Enumeration         EnumLoc;
        String30                String1Loc;
        String30                String2Loc;

#define LOOPS           50000
long                    time();
```

(continued on next page)

HIGH SPEED THRILLS

Listing Two (Listing continued, text begins on page 46.)

```
long                        starttime;
long                        benchtime;
long                        nulltime;
register unsigned int       i;
starttime = time(0);
for (i = 0; i < LOOPS; ++i);
nulltime = time(0) - starttime;

        PtrGlobNext = (RecordPtr) malloc(sizeof(RecordType));
        PtrGlob = (RecordPtr) malloc(sizeof(RecordType));
        PtrGlob->PtrComp = PtrGlobNext;
        PtrGlob->Discr = Ident1;
        PtrGlob->EnumComp = Ident3;
        PtrGlob->IntComp = 40;
        strcpy(PtrGlob->StringComp, "DHRYSTONE PROGRAM, SOME STRING");

/*****************
-- Start Timer --
*****************/
starttime = time(0);
for (i = 0; i < LOOPS; ++i)
{

        Proc5();
        Proc4();
        IntLoc1 = 2;
        IntLoc2 = 3;
        strcpy(String2Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
        EnumLoc = Ident2;
        BoolGlob = ! Func2(String1Loc, String2Loc);
        while (IntLoc1 < IntLoc2)
        {
                IntLoc3 = 5 * IntLoc1 - IntLoc2;
                Proc7(IntLoc1, IntLoc2, &IntLoc3);
                ++IntLoc1;
        }
        Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
        Proc1(PtrGlob);
        for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)
                if (EnumLoc == Func1(CharIndex, 'C'))
                        Proc6(Ident1, &EnumLoc);
        IntLoc3 = IntLoc2 * IntLoc1;
        IntLoc2 = IntLoc3 / IntLoc1;
        IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
        Proc2(&IntLoc1);

}

/*****************
-- Stop Timer --
*****************/
benchtime = time(0) - starttime - nulltime;
printf("Dhrystone time for %ld passes = %ld\n", (long) LOOPS, benchtime);
printf("This machine benchmarks at %ld dhrystones/second\n",
        ((long) LOOPS) / benchtime);

}

Proc1(PtrParIn)
REG RecordPtr   PtrParIn;
{
#define NextRecord      (*(PtrParIn->PtrComp))

        structassign(NextRecord, *PtrGlob);
        PtrParIn->IntComp = 5;
        NextRecord.IntComp = PtrParIn->IntComp;
        NextRecord.PtrComp = PtrParIn->PtrComp;
        Proc3(NextRecord.PtrComp);
        if (NextRecord.Discr == Ident1)
        {
                NextRecord.IntComp = 6;
                Proc6(PtrParIn->EnumComp, &NextRecord.EnumComp);
                NextRecord.PtrComp = PtrGlob->PtrComp;
                Proc7(NextRecord.IntComp, 10, &NextRecord.IntComp);
        }
        else
                structassign(*PtrParIn, NextRecord);

#undef  NextRecord
}

Proc2(IntParIO)
OneToFifty      *IntParIO;
{
        REG OneToFifty          IntLoc;
        REG Enumeration         EnumLoc;

        IntLoc = *IntParIO + 10;
        for(;;)
        {
                if (Char1Glob == 'A')
                {
```

```
                        --IntLoc;
                        *IntParIO = IntLoc - IntGlob;
                        EnumLoc = Ident1;
                }
                if (EnumLoc == Ident1)
                        break;
        }
}

Proc3(PtrParOut)
RecordPtr       *PtrParOut;
{
        if (PtrGlob != NULL)
                *PtrParOut = PtrGlob->PtrComp;
        else
                IntGlob = 100;
        Proc7(10, IntGlob, &PtrGlob->IntComp);
}

Proc4()
{
        REG boolean     BoolLoc;

        BoolLoc = Char1Glob == 'A';
        BoolLoc |= BoolGlob;
        Char2Glob = 'B';
}

Proc5()
{
        Char1Glob = 'A';
        BoolGlob = FALSE;
}

extern boolean Func3();

Proc6(EnumParIn, EnumParOut)
REG Enumeration EnumParIn;
REG Enumeration *EnumParOut;
{
        *EnumParOut = EnumParIn;
        if (! Func3(EnumParIn) )
                *EnumParOut = Ident4;
        switch (EnumParIn)
        {
        case Ident1:    *EnumParOut = Ident1; break;
        case Ident2:    if (IntGlob > 100) *EnumParOut = Ident1;
                        else *EnumParOut = Ident4;
                        break;
        case Ident3:    *EnumParOut = Ident2; break;
        case Ident4:    break;
        case Ident5:    *EnumParOut = Ident3;
        }
}

Proc7(IntParI1, IntParI2, IntParOut)
OneToFifty      IntParI1;
OneToFifty      IntParI2;
OneToFifty      *IntParOut;
{
        REG OneToFifty  IntLoc;

        IntLoc = IntParI1 + 2;
        *IntParOut = IntParI2 + IntLoc;
}

Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim       Array1Par;
Array2Dim       Array2Par;
OneToFifty      IntParI1;
OneToFifty      IntParI2;
{
        REG OneToFifty  IntLoc;
        REG OneToFifty  IntIndex;

        IntLoc = IntParI1 + 5;
        Array1Par[IntLoc] = IntParI2;
        Array1Par[IntLoc+1] = Array1Par[IntLoc];
        Array1Par[IntLoc+30] = IntLoc;
        for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
                Array2Par[IntLoc][IntIndex] = IntLoc;
        ++Array2Par[IntLoc][IntLoc-1];
        Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
        IntGlob = 5;
}

Enumeration Func1(CharPar1, CharPar2)
CapitalLetter   CharPar1;
CapitalLetter   CharPar2;
{
        REG CapitalLetter       CharLoc1;
        REG CapitalLetter       CharLoc2;
```

(continued on next page)

Listing Two (Listing continued, text begins on page 46.)

```
        CharLoc1 = CharPar1;
        CharLoc2 = CharLoc1;
        if (CharLoc2 != CharPar2)
                return (Ident1);
        else
                return (Ident2);
}

boolean Func2(StrParI1, StrParI2)
String30        StrParI1;
String30        StrParI2;
{
        REG OneToThirty         IntLoc;
        REG CapitalLetter       CharLoc;

        IntLoc = 1;
        while (IntLoc <= 1)
                if (Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1)
                {
                        CharLoc = 'A';
                        ++IntLoc;
                }
        if (CharLoc >= 'W' && CharLoc <= 'Z')
                IntLoc = 7;
        if (CharLoc == 'X')
                return(TRUE);
        else
        {
                if (strcmp(StrParI1, StrParI2) > 0)
                {
                        IntLoc += 7;
                        return (TRUE);
                }
                else
                        return (FALSE);
```

```
        }
}

boolean Func3(EnumParIn)
REG Enumeration EnumParIn;
{
        REG Enumeration EnumLoc;

        EnumLoc = EnumParIn;
        if (EnumLoc == Ident3) return (TRUE);
        return (FALSE);
}

#ifdef  NOSTRUCTASSIGN
memcpy(d, s, l)
register char   *d;
register char   *s;
int     l;
{
        while (l--) *d++ = *s++;
}
#endif

/*
 *      Library function for compilers with no time(2) function in the
 *      library.
 */
#ifdef  NOTIME
long    time(p)
long    *p;
{               /* CI-C86 time function - don't use around midnight */
        long    t;
        struct regval {unsigned int ax,bx,cx,dx,si,di,ds,es; } regs;

        regs.ax = 0x2c00;
        sysint21(&regs, &regs);
        t = ((regs.cx>>8)*60L + (regs.cx & 0xff))*60L + (regs.dx>>8);
        if (p) *p = t;
        return t;
}
#endif
```

End Listings

THE PROGRAMMER'S SHOP

provides complete information, advice, guarantees and every product for Microcomputer Programming.

GREENLEAF BARGAINS ORDER TODAY

Order before 9/31/86 and mention this ad for the special prices to the right:

Greenleaf Functions - 200+ functions in C for printer, formatting. With source. Covers all bases including screen, graphics, & DOS interface. Good doc. Specify compiler.
Greenleaf Comm Library - Full ASYNCH support with source. Xmodem, CRC, polling. Specify compiler.

	LIST	Before 9/30	After 9/30
Greenleaf Functions	185	119	139
Greenleaf Communications	185	119	139

C Support-Systems

Basic-C Library by C Source PC $139
C Sharp - well supported. Source, **realtime**, tasks, state system MS $600
C ToolSet - DIFF, xref, source MS $ 95
The HAMMER by OES Systems PC $179
Lattice Text Utilities MS $105
PC LINT - Checker. Amiga $89 MS $119
SECURITY LIB - add encrypt to MS C, C86 programs. Source $250 PC $125

C-Screens, Windows, Graphics

C Power Windows by Entelekon PC $119
dBASE Graphics for C PC $ 79
Curses by Lattice PC $109
ESSENTIAL GRAPHICS - fast, fonts, no royalties PC $219
GraphiC - new color version PC $319
Topview Toolbasket by Lattice PC $209
View Manager for C by Blaise PC $219
Vitamin C - screen I/O PC $139
Windows for C - fast PC $159
Windows for Data - validation PC $239

Debuggers

Advanced Trace-86 by Morgan Modify ASM code on fly. PC $149
CODESMITH - visual, modify and rewrite Assembler PC $109
C SPRITE - data structures PC $139
DSD87 - windowing, 8087 MS $ 95
Periscope I - own 16K PC $269
Periscope II - symbolic, "Reset Box," 2 Screen PC $129
Pfix-86 Plus Symbolic Debugger by Phoenix - windows PC $249
Software Source by Atron - Lattice, MS C, Pascal, Windows single step, 2 screen, log file. MS $115
w/Breakswitch $199

FEATURE

Smalltalk-80 - "Official" Xerox licensed version. Compatible with Smalltalk-80 v2 running on other systems. Compiler, complete graphics interface, debugger, windows, text and graphics editor, source. Protected mode, RAM access. Requires PC AT PC $995

Fortran & Supporting

ACS Time Series MS $449
Forlib+ by Alpha - graphics and file routines, comm. MS $ 59
MACFortran by Microsoft MAC $229
MS Fortran link to C MS **$219**
No Limit - Fortran Scientific PC $119
PolyFortran - xref, pp, screen MS $149
Prospero - '66, reentrant MS $349
RM/Fortran - enhanced "IBM Professional Fortran" MS $395
Scientific Subroutines - Matrix MS $149
Statistician by Alpha MS $269
Strings and Things - register, shell PC $ 59

Multilanguage Support

BTRIEVE ISAM MS $199
BTRIEVE/N-multiuser MS $469
CODESIFTER - Profiler. MS $109
HALO Graphics - 115+ devices. Animation, engineering, business. Any MS language, Lattice, C86 PC $229
PANEL - Create screen with editor, generates code. Full data validation, no royalties. Xenix $539, MS $239
Pfinish Performance Analyzer PC $249
PLINK 86-program-independent MS $249
PLINK-86 PLUS - incremental MS $369
PolyLibrarian by Polytron MS $ 85
PVCS Version Control MS $329
Rtrieve - Btrieve front end MS $ 79
Screen Sculptor - slick, thorough, fast, BASIC, PASCAL. PC $ 99
Xtrieve - organize database MS $169
ZAP Communications - VT 100, TEK 4010 emulation, file xfer. PC $ 95
ZView - screen generator MS $219

Pascal and Supporting

Alice - learn Pascal. Turbo compatible, interpreter PC $ 85
MetaWINDOW - graphics toolkit PC $119
Microsoft PASCAL - faster MS $109
MICROTEC PASCAL - 5 memory models, "Iterators", 65 bit 8087 strings MS $665
Pascal Tools - strings, screen PC $109
Pascal Tools 2 - by Blaise MS $ 85
Pfas - Portable Isam MS $185
Prospero Pascal - full ISO+ MS $349
USCD Pascal - native code MS $ 69

RECENT DISCOVERY

Lattice RPG II Compiler - Run RPG II programs developed for the System III or system 32/34/36 with little or no change in source code. Screen gen, ISAM, & direct files. No royalties. PC $639

Other Languages

APL*PLUS/PC PC $ 469
Artek ADA Compiler - DOD standard minus multitasking PC $ 895
CLIPPER-dBASE Compiler MS $ 449
ED/ASM - 86 by Oliver PC $ 85
MacASM - fast MAC $ 99
MasterForth - Forth '83 MAC or PC $ 125
Microsoft MASM - faster MS $ 109
Modula 2 by Volition Systems MS $ 250
Modula-2/86 Compiler by Logitech w/ 8087 ($105), 512K ($149). PC $ 65
Pasm - by Phoenix MS $ 219
RPG II by Lattice PC $ 639
SNOBOL4+ - great for strings MS $ 85
Turbo Edit/ASM - by Speedware PC $ 85

Xenix-86 & Supporting

Basic - by Microsoft $ 279
Cobol - by Microsoft $ 795
Fortran - by Microsoft $ 399
Xenix Complete Development System. $1149

Other Products

BSW Make - like UNIX make MS $ 85
Dan Bricklin's Demo Program PC $ 65
dBrief - Customize BRIEF for dBASE development. with BRIEF $275. PC $ 95
H Test/H Format - XT Fix PC $ 89
Interactive Easyflow-HavenTree PC $ 139
Link & Locate - MSDOS tools to work with Intel and Tektronix projects. MS $ 329
LMK - like UNIX make PC $ 149
Microsoft Windows PC $ 75
Microsoft Windows Software Development Kit PC $ 399
Opt Tech Sort - sort, merge MS $ 119
PMaker - by Phoenix PC $ 139
Polymake by Polytron MS $ 79
PolyWindows Dev. Kit PC $ 149
PS MAKE MS $ 129
SECRET DISK by Lattice PC $ 99
SET:SCIL - manage revisions PC $ 299
Shrink/Shrinkem - put more files on disk PC $ 150
SoftEst - Manage projects. MS $ 350
Source Print PC $ 89
Synergy-Create user interfaces MS $ 375
Texsys - control source MS $ 89
Visible Computer: 8088 - Simulates demos or any .exe. com, Debugger. 350 pg. tutorial, unprotected PC $ 69

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and POs. All formats available. UPS surface shipping add $3/item.

Call for a catalog, literature, advice and service you can trust

HOURS
8:30 AM - 8:00 PM EST.

800-421-8006

THE PROGRAMMER'S SHOP™
128-D Rockland Street, Hanover, MA 02339
Mass: 800-442-8070 or 617-826-7531 7/86

"You've got everything I've heard of, and much I haven't! . . . Normally, I expect my money to be "fan letter" enough, but you people are SUPER!"

Shel Hall
Artell Corp.

Listing One (Text begins on page 108.)

```
Listing 1.


1) copy masm.exe masm.sav (to make a backup)

2) ren masm.exe masm.fix (debug.com can't write EXE files)

3) debug masm.fix
  3a) D 72B8 L 22   (should see 34 bytes of 00)
  3b) E 72B8
      enter these bytes:
      8B 1E D6 09 C6 06 C0 01 00 E9 75 02 C4 57 06 FE
      06 C0 01 E9 74 02 80 7C FF 1A 74 03 39 9D 02 4E
      EB F4
  3c) U 7535 L 1 (should see "MOV BX,[09D6]" )
  3d) E 7535
      enter these bytes:  E9 80 FD
  3e) U 753F L 1  (should see "LES DX,[BX+06]" )
  3f) E 753F
      enter these bytes:  E9 82 FD
  3g) U 756D L 1  (should see "CMP BYTE PTR [SI-01],1A" )
      enter these bytes:  E9 5E FD
  3h) W  (to write changes to masm.fix)
  3i) Q  (to exit debug.com)

4) ren masm.fix masm.exe

5) Test masm.exe to make sure changes have been made correctly.
   If not, copy masm.sav to masm.exe and try again.
```

End Listing One

Listing Two

```
Listing Two

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;                                                                      ;;;;
;;;;      hpkio        Basic I/O functions using handle packing.          ;;;;
;;;;                                                                      ;;;;
;;;;              Written By    Paul M. Adams                             ;;;;
;;;;                            Route 4 Box 23                            ;;;;
;;;;                            Shelbyville, KY  40065                    ;;;;
;;;;                                                                      ;;;;
;;;;                                                                      ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


        include model.h
        include prologue.h


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;     hcreate(dsn, attr)                                                 ;;;
;;;     unsigned char    *dsn;    /* Data Set Name            */           ;;;
;;;     int              attr;    /* file attribute byte      */           ;;;
;;;                                                                        ;;;
;;;                                                                        ;;;
;;;     hold_handle = psp_handle[19]                                       ;;;
;;;     psp_handle[19] = FF                                                ;;;
;;;     call dos  to create file                                           ;;;
;;;     if error                                                           ;;;
;;;             retval = -1                                                ;;;
;;;     else                                                               ;;;
;;;             retval =  psp_handle[dos_handle]                           ;;;
;;;             psp_handle[dos_handle] = FF                                ;;;
;;;                                                                        ;;;
;;;     psp_handle[19] = hold_handle                                       ;;;
;;;                                                                        ;;;
;;;     return(retval)   - real dos handle                                 ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

hcreate proc
        public  hcreate
        push    es
        push    bp
        mov     bp,sp
        sub     sp, 2
dsn          equ [bp + 6]
attr         equ [bp + 8]
hold_handle  equ [bp - 2]

        mov     ah, 62h
        int     21h
        mov     es, bx
```

Product Information

Free!

Dr. Dobb's Journal of Software Tools

September 1986 #119 **Expiration Date:** **December 31, 1986**

Name ______________________ Title ______________

Company ______________________ Phone ______________

Address ______________________________________

City/State/Zip ______________________________

Please circle one letter in each category:

I. My work is performed:
A. for in-house use only.
B. for other companies.
C. for end users/retailers.
D. in none of the above areas.

II. My primary job function:
A. Software Project Mgmt/Spvr
B. Hardware Project Mgmt/Spvr
C. Computer Consultant
D. Corporate Management
E. Other

III. My company department performs:
A. software development.
B. computer system integration.
C. computer manufacturing.
D. computer consulting.
E. computer research.
F. none of the above.

IV. This inquiry is for:
A. a purchase within 1 month.
B. a purchase within 1 to 6 months.
C. product information only.

V. Corporate Purchase Authority:
A. Final Decision-maker
B. Approve/Recommend
C. No Influence

VI. Personal Computer Users at my Jobsite:
A. 10,000 or more
B. 500 to 9,999
C. 100 to 499
D. 10 to 99
E. less than 10

VII. On average, I advise others about computers:
A. more than once per day.
B. once per day.
C. once per week.
D. less than once per week.

VIII. In my job function, I:
A. design software and/or write code.
B. design software.
C. write code.
D. don't design software or write code.

A Reader Service number appears on each advertisement. Circle the corresponding numbers below for more info.

001	002	003	004	005	006	007	008	009
010	011	012	013	014	015	016	017	018
019	020	021	022	023	024	025	026	027
028	029	030	031	032	033	034	035	036
037	038	039	040	041	042	043	044	045
046	047	048	049	050	051	052	053	054
055	056	057	058	059	060	061	062	063
064	065	066	067	068	069	070	071	072
073	074	075	076	077	078	079	080	081
082	083	084	085	086	087	088	089	090
091	092	093	094	095	096	097	098	099
100	101	102	103	104	105	106	107	108
109	110	111	112	113	114	115	116	117
118	119	120	121	122	123	124	125	126
127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153
154	155	156	157	158	159	160	161	162
163	164	165	166	167	168	169	170	171
172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189
190	191	192	193	194	195	196	197	198
199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216
217	218	219	220	221	222	223	224	225
226	227	228	229	230	231	232	233	234
235	236	237	238	239	240	241	242	243
244	245	246	247	248	249	250	251	252
253	254	255	256	257	258	259	260	261
262	263	264	265	266	267	268	269	270
271	272	273	274	275	276	277	278	279
280	281	282	283	284	285	286	287	288
289	290	291	292	293	294	295	296	297
298	299	999						

Circle 999 to start a 12 month subscription at the price of $29.97

Thank You!
Dr. Dobb's greatly appreciates your responses to questions I through VIII.

Postage Paid!

Product Information

Free!

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
First Class Permit #217, Clinton, Iowa

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER
P.O. Box 2157
Clinton, Iowa 52735-2157

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
First Class Permit #217, Clinton, Iowa

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER
P.O. Box 2157
Clinton, Iowa 52735-2157

Product Information

Free!

Thank You!
Dr. Dobb's greatly appreciates your responses to questions I through VIII.

Postage Paid!

A Reader Service number appears on each advertisement. Circle the corresponding numbers below for more info.

001	002	003	004	005	006	007	008	009
010	011	012	013	014	015	016	017	018
019	020	021	022	023	024	025	026	027
028	029	030	031	032	033	034	035	036
037	038	039	040	041	042	043	044	045
046	047	048	049	050	051	052	053	054
055	056	057	058	059	060	061	062	063
064	065	066	067	068	069	070	071	072
073	074	075	076	077	078	079	080	081
082	083	084	085	086	087	088	089	090
091	092	093	094	095	096	097	098	099
100	101	102	103	104	105	106	107	108
109	110	111	112	113	114	115	116	117
118	119	120	121	122	123	124	125	126
127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153
154	155	156	157	158	159	160	161	162
163	164	165	166	167	168	169	170	171
172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189
190	191	192	193	194	195	196	197	198
199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216
217	218	219	220	221	222	223	224	225
226	227	228	229	230	231	232	233	234
235	236	237	238	239	240	241	242	243
244	245	246	247	248	249	250	251	252
253	254	255	256	257	258	259	260	261
262	263	264	265	266	267	268	269	270
271	272	273	274	275	276	277	278	279
280	281	282	283	284	285	286	287	288
289	290	291	292	293	294	295	296	297
298	299	999						

Circle 999 to start a 12 month subscription at the price of $29.97

Dr. Dobb's Journal of Software Tools

September 1986 #119 **Expiration Date:** December 31, 1986

Name ______________________ Title ______________

Company ____________________ Phone ______________

Address ______________________________________

City/State/Zip _________________________________

Please circle one letter in each category:

I. My work is performed:
A. for in-house use only.
B. for other companies.
C. for end users/retailers.
D. in none of the above areas.

II. My primary job function:
A. Software Project Mgmt/Spvr
B. Hardware Project Mgmt/Spvr
C. Computer Consultant
D. Corporate Management
E. Other

III. My company department performs:
A. software development.
B. computer system integration.
C. computer manufacturing.
D. computer consulting.
E. computer research.
F. none of the above.

IV. This inquiry is for:
A. a purchase within 1 month.
B. a purchase within 1 to 6 months.
C. product information only.

V. Corporate Purchase Authority:
A. Final Decision-maker
B. Approve/Recommend
C. No Influence

VI. Personal Computer Users at my Jobsite:
A. 10,000 or more
B. 500 to 9,999
C. 100 to 499
D. 10 to 99
E. less than 10

VII. On average, I advise others about computers:
A. more than once per day.
B. once per day.
C. once per week.
D. less than once per week.

VIII. In my job function, I:
A. design software and/or write code.
B. design software.
C. write code.
D. don't design software or write code.

Product Information

Free!

```
        mov     al, byte ptr es:[18h + 19]
        xor     ah, ah
        mov     hold_handle, ax
        mov     byte ptr es:[18h + 19], 0ffh

        mov     ah, 3ch
        mov     dx, dsn
        mov     cx, attr
        int     21h
        jc      hcreate_error

        mov     bx, ax
        mov     al, byte ptr es:[bx + 18h]
        xor     ah, ah
        mov     byte ptr es:[bx + 18h], 0ffh
        jmp     hcreate_done

hcreate_error:
        mov     ax, 0ffffh

hcreate_done:
        push    ax
        mov     ax, hold_handle
        mov     byte ptr es:[18h + 19], al
        pop     ax

hcreate_exit:
        mov     sp, bp
        pop     bp
        pop     es
        ret

hcreate endp


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                     ;;;
;;;     hopen(dsn, mode)                                                ;;;
;;;     unsigned char   *dsn;   /* Data Set Name          */           ;;;
;;;     int             mode;   /* DOS open mode          */           ;;;
;;;                                                                     ;;;
;;;     hold_handle = psp_handle[19]                                    ;;;
;;;     psp_handle[19] = FF                                             ;;;
;;;     call dos  to open file                                          ;;;
;;;     if error                                                        ;;;
;;;             retval = -1                                             ;;;
;;;     else                                                            ;;;
;;;             retval =  psp_handle[dos_handle]                        ;;;
;;;             psp_handle[dos_handle] = FF                             ;;;
;;;                                                                     ;;;
;;;     psp_handle[19] = hold_handle                                    ;;;
;;;                                                                     ;;;
;;;     return(retval)   - real dos handle                              ;;;
;;;                                                                     ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

hopen   proc
        public  hopen

        push    es
        push    bp
        mov     bp,sp
        sub     sp, 2

dsn             equ [bp + 6]
mode            equ [bp + 8]
hold_handle equ [bp - 2]

        mov     ah, 62h
        int     21h
        mov     es, bx

        mov     al, byte ptr es:[18h + 19]
        xor     ah, ah
        mov     hold_handle, ax
        mov     byte ptr es:[18h + 19], 0ffh

        mov     ax, mode
        mov     ah, 3dh
        mov     dx, dsn
        int     21h
        jc      hopen_error

        mov     bx, ax
        mov     al, byte ptr es:[bx + 18h]
        xor     ah, ah
        mov     byte ptr es:[bx + 18h], 0ffh
        jmp     hopen_done

hopen_error:
        mov     ax, 0ffffh

hopen_done:
        push    ax
        mov     ax, hold_handle
        mov     byte ptr es:[18h + 19], al
        pop     ax

hopen_exit:
```

(continued on next page)

Listing Two (Listing continued, text begins on page 108.)

```
        mov     sp, bp
        pop     bp
        pop     es
        ret
hopen   endp


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                         ;;;
;;;     hclose(handle)                                                      ;;;
;;;     int             handle;         /* File handle from hopen   */     ;;;
;;;                                                                         ;;;
;;;                                                                         ;;;
;;;     hold_handle = psp_handle[19]                                        ;;;
;;;     psp_handle[19] = handle                                             ;;;
;;;                                                                         ;;;
;;;     call dos  to close handle                                           ;;;
;;;     if error                                                            ;;;
;;;             retval = -1                                                 ;;;
;;;     else                                                                ;;;
;;;             retval = 0                                                  ;;;
;;;                                                                         ;;;
;;;     psp_handle[19] = hold_handle                                        ;;;
;;;     return (retval)                                                     ;;;
;;;                                                                         ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


hclose  proc
        public  hclose

        push    es
        push    bp
        mov     bp,sp
        sub     sp, 2

handle      equ [bp + 6]
hold_handle equ [bp - 2]

        mov     ah, 62h
        int     21h
        mov     es, bx

        mov     al, byte ptr es:[18h + 19]
        xor     ah, ah
        mov     hold_handle, ax
        mov     ax, handle
        mov     byte ptr es:[18h + 19], al
        mov     bx, 19
        mov     ah, 3eh
        int     21h
        jc      hclose_error

        xor     ax, ax
        jmp     hclose_done

hclose_error:
        mov     ax, 0ffffh

hclose_done:
        push    ax
        mov     ax, hold_handle
        mov     byte ptr es:[18h + 19], al
        pop     ax

hclose_exit:
        mov     sp, bp
        pop     bp
        pop     es
        ret

hclose  endp


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                         ;;;
;;;     hget(handle, ioarea, len)                                           ;;;
;;;     int             handle;         /* File handle from hopen   */     ;;;
;;;     unsigned char   *ioarea;        /* Input Buffer             */     ;;;
;;;     int             len;            /* Number of bytes to read  */     ;;;
;;;                                                                         ;;;
;;;     hold_handle = psp_handle[19]                                        ;;;
;;;     psp_handle[19] = handle                                             ;;;
;;;                                                                         ;;;
;;;     call dos  to read file                                              ;;;
;;;     if error                                                            ;;;
;;;             retval = -1                                                 ;;;
;;;     else                                                                ;;;
;;;             retval = number of bytes read                               ;;;
;;;                                                                         ;;;
;;;     psp_handle[19] = hold_handle                                        ;;;
;;;     return (retval)                                                     ;;;
;;;                                                                         ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
hget    proc
        public  hget

        push    es
        push    bp
        mov     bp,sp
        sub     sp, 2

handle       equ [bp + 6]
get_area     equ [bp + 8]
get_len      equ [bp + 10]
hold_handle  equ [bp - 2]

        mov     ah, 62h
        int     21h
        mov     es, bx

        mov     al, byte ptr es:[18h + 19]
        xor     ah, ah
        mov     hold_handle, ax

        mov     ax, handle
        mov     byte ptr es:[18h + 19], al
        mov     dx, get_area
        mov     cx, get_len
        mov     bx, 19
        mov     ah, 3fh
        int     21h
        jc      hget_error
        jmp     hget_done

hget_error:
        mov     ax, 0ffffh

hget_done:
        push    ax
        mov     ax, hold_handle
        mov     byte ptr es:[18h + 19], al
        pop     ax

hget_exit:
        mov     sp, bp
        pop     bp
        pop     es
        ret

hget    endp


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                     ;;;
;;;     hput(handle, ioarea, len)                                       ;;;
;;;     int             handle;         /* File handle from hopen  */   ;;;
;;;     unsigned char   *ioarea;        /* Output buffer           */   ;;;
;;;     int             len;            /* Number of bytes to write*/   ;;;
;;;                                                                     ;;;
;;;     hold_handle = psp_handle[19]                                    ;;;
;;;     psp_handle[19] = handle                                         ;;;
;;;                                                                     ;;;
;;;     call dos  to write file                                         ;;;
;;;     if error                                                        ;;;
;;;             retval = -1                                             ;;;
;;;     else                                                            ;;;
;;;             retval = number of bytes written                        ;;;
;;;                                                                     ;;;
;;;     psp_handle[19] = hold_handle                                    ;;;
;;;     return (retval)                                                 ;;;
;;;                                                                     ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
hput    proc
        public  hput

        push    es
        push    bp
        mov     bp,sp
        sub     sp, 2

handle       equ [bp + 6]
put_area     equ [bp + 8]
put_len      equ [bp + 10]
hold_handle  equ [bp - 2]

        mov     ah, 62h
        int     21h
        mov     es, bx

        mov     al, byte ptr es:[18h + 19]
        xor     ah, ah
        mov     hold_handle, ax

        mov     ax, handle
        mov     byte ptr es:[18h + 19], al
        mov     dx, put_area
        mov     cx, put_len
        mov     bx, 19
        mov     ah, 40h
        int     21h
        jc      hput_error
        jmp     hput_done
```

(continued on next page)

Listing Two (Listing continued, text begins on page 108.)

```
hput_error:
        mov     ax, 0ffffh

hput_done:
        push    ax
        mov     ax, hold_handle
        mov     byte ptr es:[18h + 19], al
        pop     ax

hput_exit:
        mov     sp, bp
        pop     bp
        pop     es
        ret

hput    endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                   ;;;
;;;     long int hseek(handle, rba, method)                           ;;;
;;;     int             handle;         /* File handle from hopen  */ ;;;
;;;     long int        rba;            /* Relative Byte address   */ ;;;
;;;     int             method;         /* Seek method             */ ;;;
;;;                                                                   ;;;
;;;     hold_handle = psp_handle[19]                                  ;;;
;;;     psp_handle[19] = handle                                       ;;;
;;;                                                                   ;;;
;;;     call dos  to seek to rba                                      ;;;
;;;     if error                                                      ;;;
;;;             retval = -1                                           ;;;
;;;     else                                                          ;;;
;;;             retval = seek address                                 ;;;
;;;                                                                   ;;;
;;;     psp_handle[19] = hold_handle                                  ;;;
;;;     return (retval)                                               ;;;
;;;                                                                   ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

hseek   proc
        public  hseek

        push    es
        push    bp
        mov     bp,sp
        sub     sp, 2

handle      equ [bp + 6]
rba_low     equ [bp + 8]
rba_high    equ [bp + 10]
seek_method equ [bp + 12]
hold_handle equ [bp - 2]

        mov     ah, 62h
        int     21h
        mov     es, bx

        mov     al, byte ptr es:[18h + 19]
        xor     ah, ah
        mov     hold handle, ax
        mov     ax, handle
        mov     byte ptr es:[18h + 19], al
        mov     dx, rba_low
        mov     cx, rba_high
        mov     ax, seek_method
        mov     ah, 42h
        mov     bx, 19
        int     21h
        jc      hseek_error
        jmp     hseek_done

hseek_error:
        mov     ax, 0ffffh
        mov     dx, ax

hseek_done:
        push    ax
        mov     ax, hold_handle
        mov     byte ptr es:[18h + 19], al
        pop     ax

hseek_exit:
        mov     sp, bp
        pop     bp
        pop     es
        ret
hseek   endp

        include epilogue.h
        end
```

End Listing Two

Listing Three

```
Listing Three

#include <stdio.h>

#define MAXF  256

struct  { int   scs, sss, sds, ses; }  sreg;
struct  { int   ax, bx, cx, dx, si, di, ds, es; }    dreg;

main(argc, argv)
int             argc;
unsigned char   *argv;
{
        int     fid[MAXF];
        int     i;
        int     j;
        int     c;
        long    l;
        int     lastfile;
        char    dsn[30];
        char    line[80];
        long int        hseek();


        segread(&sreg);
        dreg.ax = 0x6200;
        sysint21(&dreg, &dreg);

        printf("\nCreate files");

        for (i = 0; i < MAXF ; i++ ) {
                sprintf(dsn, "\\test\\bt%d.dat", i);
                c = hcreate(dsn, 0);
                fid[i] = c;
                printf("\nCreate dsn = %s, handle = %3d", dsn, c);

                if (c == -1) {
                        break;
                }
        }
        lastfile = i;

        printf("\n");

        printf("\nWrite files");

        for (i = 0; i < lastfile; i++ ) {
            printf("\nWriting file number %02d", i);

            for(j = 0; j < 10; j++ ) {
                sprintf(line, "out file no %03d line no %02d\n\r", i, j);
                c = hput(fid[i], line, 28);
            }
        }
       printf("\nClose files");

       for (i = 0; i < lastfile; i++ ) {
               c = hclose(fid[i]);
               printf("\nreturn from close %3d = %3d", i, c );

       }

      printf("\nOpen files");

      for (i = 0; i < lastfile; i++ ) {
              sprintf(dsn, "\\test\\bt%d.dat", i);
              c = hopen(dsn, 0);
              fid[i] = c;
              printf("\nOpen dsn = %s, handle = %3d", dsn, c);

      }

      printf("\nRead files");

      for (i = 0; i < lastfile; i++ ) {
              l = 28 * (i % 10);
              l = hseek(fid[i], l, 0);
              c = hget(fid[i], line, 28);
              line[26] = 0;
              printf("\nseek return = %3D, get return = %3d, line = %s", l, c, line);

      }

      printf("\nClose files");

      for (i = 0; i < lastfile; i++ ) {
              c = hclose(fid[i]);
              printf("\nreturn from close %3d = %3d", i, c );

      }
}
```

End Listings

Listing One (Text begins on page 114.)

```
                      INFO     $Header: worm.a-v 1.2 86/03/24 01:44:36 jans Exp $
                 **** The Worm Memory Test          ****************************************
                 * Author: Jan W. Steinman, 2002 Parkside Ct.,  West Linn, OR 97068.
                 *
                 * The Worm memory test has three parts.  Init sets up the registers for the
                 * Worm.  The Display Manager interacts with the Worm each pass and periodically
                 * Displays the Worm's progress.  The Worm itself Worms itself through memory,
                 * from high to low, checking memory against a copy of itself.  The Droppings
                 * form a pattern through memory when the test is complete.
                 *
                 * This version runs on the Tektronix 4404 under Uniflex.  System dependent code
                 * is mostly segregated to the Init, Display, Disable and Enable routines.  Two
                 * instructions in the Worm routine are system dependent, for enabling and
                 * disabling interrupts.
                 *
                 * Register usage:
                 *     D0        scratch register.
                 *     D1        scratch register.
                 *     D2        scratch register.
                 *     D3        scratch register.
                 *     D4
                 *     D5        address mask for determining if time to show progress.
                 *     D6        base of memory area under test.
                 *     D7        length of Worm in long words.
                 *     A0        scratch register.
                 *     A1        scratch register.
                 *     A2        scratch register.
                 *     A3        pointer to Display manager for position independent access.
                 *     A4        pointer to permanent Worm image for comparison.
                 *     A5        pointer to crawling Worm image.
                 *     A6
                 *     A7        stack pointer.
                 *
                 * These included files contain system definitions and interrupt (signal)
                 * numbers for the Uniflex operating system.  Don't bother to list these.
                 *
                      OPT      lis
                      DEFINE                    (This makes all labels global for debug.)
                 *
                 * Set D_MASK with the bits that are zero at each progress report.
                 *
        0000 03FC D_MASK   EQU      $00003FC      Report each boundary passed.
        0000 0004 REL_SIZ  EQU      4             Relocation is four bytes at a time.
        0000 8000 MEM_SIZ  EQU      $2000*REL_SIZ Test a 32K chunk.
        0000 0002 DISABLE  EQU      2             Trap number for Disable routine.
        0000 0003 ENABLE   EQU      3             Trap number for Enable routine.
        0000 000D CR       EQU      $0D           Carriage return.
        0000 000A LF       EQU      $0A           Line feed.
                 *
                 * Uniflex will not allow intersection math, so put all the code in the DATA
                 * section, and don't use TEXT or BSS at all!
                 *
000000                DATA                      Assemble into writable data section.
        0000 0000 MemBeg   EQU      *

                 **** hexadecimalize     ****************************************************
                 * hexadecimalize converts a long word to eight ASCII hexadecimal characters.
                 * This routine is machine and OS independent.  It uses a simple table look-up
                 * to generate the hexadecimal string.
                 *
                 *     Entry:   d0 -- Long word to be converted to hex.
                 *              a0 -- Pointer to buffer where hex characters will go.
                 *
                 *     Exit:    d2 -- -1.  (Just in case someone cares!)
                 *              d0 -- unchanged.
                 *              -8(a0) -- points to eight ASCII characters.
                 *
                 *     Uses:    d3 -- nybble mask: constant $0F.
                 *              d2 -- nybble counter.
                 *              d1 -- current nybble to convert is LSN.
                 *
000000 3031 3233 3435 CharTab  DC.B    '0123456789ABCDEF' Where we keep our hex characters.
000010                hexadecimalize
000010 7407                    move.l  #7,d2         Bytes to make - 1.
000012 760F                    move.l  #$0F,d3       Nybble mask.
000014 E998           HexLoop  rol.l   #4,d0         Shift the next nybble into the LSN,    <--------+
000016 2200                    move.l  d0,d1              make a copy for masking,                   |
000018 C283                    and.l   d3,d1              mask out all but least significant nybble, |
                      *                                   index into char table and store result.    |
00001A 10FB 10E4               move.b  CharTab(pc,d1),(a0)+                                          |
00001E 51CA FFF4               dbra    d2,HexLoop    Repeat until done, and when done,     -----------+
000022 4E75                    rts                        hit the road, Jack. -->

                      **** Manager  ****************************************************************
                      * Manager checks the Worm's progress, and periodically reports to the Display.
                      * This routine is also entered if an error is encountered.
                      *
                      *     Entry:   d0 -- W_LONGS complement of pass count if error, else -1.
                      *
                      *              a1 -- test address pass/fail value.
                      *
                      *     Exit:    via direct jump to Worm at (A5).
                      *
```

```
                    *     Uses:     d3, d2, d1, d0, a7, a1, a0
                    *
                    *     Stack:    one level, plus needs of Display.
                    *
000024 0D57 6F72 6D20 ErrMsg    DC.B    CR,'Worm reports memory error at '
000042                ErrAddrMsg
000042 3030 3030 3030           DC.B    '00000000 on pass '
000053                ErrCountMsg
000053 3030 3030 3030           DC.B    '00000000.',CR
       0000 0039      E_SIZ     EQU     *-ErrMsg
00005D 0D57 6F72 6D20 DoneMsg   DC.B    CR,'Worm tested memory from '
000076                DoneBegAddrMsg
000076 3030 3030 3030           DC.B    '00000000 through '
000087                DoneEndAddrMsg
000087 3030 3030 3030           DC.B    '00000000 successfully.',CR
       0000 0041      D_SIZ     EQU     *-DoneMsg
00009E 3030 3030 3030 ProgMsg   DC.B    '00000000',CR
       0000 0009      P_SIZ     EQU     *-ProgMsg
0000A8 00                       EVEN                   (Stay on legal instruction boundary.)
0000A8 4A40           Manager   tst.w   d0             Was loop exited by error, or countdown?
0000AA 6A30                     bpl.s   GetErrMsg        Error, go report it. -----------------------+
0000AC BC8D                     cmp.l   a5,d6            Countdown, so are we done yet?             |
0000AE 6708                     beq.s   GetDoneMsg         Yes.  Go finish up. --------------------+ |
0000B0 200D                     move.l  a5,d0              No, put the new source where we can    | |
0000B2 C085                     and.l   d5,d0                look at the bottom bits: on boundary? | |
0000B4 674A                     beq.s   Report                 Yes, set up for progress report. ---|+|
0000B6 4ED5                     jmp     (a5)                   No.  Keep on Crawlin'... -->        |||
                    *                                   Finish up.  Get the pointer to start addr, |||
0000B8 41Fa FFBC      GetDoneMsg lea    DoneBegAddrMsg(pc),a0   <------------------------------------+||
0000BC 2009                     move.l  a1,d0            and the value to plug in,                  ||
0000BE 6100 FF50                bsr     hexadecimalize   which gets converted, likewise, get        ||
0000C2 41FA FFC3                lea     DoneEndAddrMsg(pc),a0                                        ||
0000C6 203C 0000 8000           move.l  #MEM_SIZ,d0      the end address and its value,             ||
0000CC 6100 FF42                bsr     hexadecimalize   also converted to hexAscii.                ||
0000D0 41FA FF8B                lea     DoneMsg(pc),a0 Get pointer to complete done message,        ||
0000D4 7641                     move.l  #D_SIZ,d3        length of the done message,                ||
0000D6 487A 0050                pea     Exit(pc)         push a return pointer,                     ||
0000DA 6034                     bra.s   Display          and go display the message. --------------+||
                    *                                Make an error report.    Get message ptr,      |||
0000DC 41FA FF75      GetErrMsg lea     ErrCountMsg(pc),a0  <-----------------------------------------||+
0000E0 0400 0007                sub.b   #W_LONGS-1,d0    convert worm count to a pass count,        ||
0000E4 6100 FF2A                bsr     hexadecimalize   make it hex for Display. <-->              ||
                    *                                Get addr of ASCII error addr,                  ||
0000E8 41FA FF58                lea     ErrAddrMsg(pc),a0                                            ||
```

(continued on next page)

Listing One (Listing continued, text begins on page 114.)

```
 0000EC 70FC               move.l  #-4,d0             get bad long addr to display,            ||
 0000EE D089               add.l    a1,d0             less four to account for postincrement,  ||
The Worm Memory Test                    Mon Mar 24 02:15:36 1986    page 4


 0000F0 6100 FF1E          bsr     hexadecimalize     make it hex for Display. <-->            ||
 0000F4 41FA FF2E          lea      ErrMsg(pc),a0    Get pointer to whole err msg,             ||
 0000F8 7639               move.l  #E_SIZ,d3          the size for the write,                  ||
 0000FA 487A 002C          pea      Exit(pc)          push a return pointer,                   ||
 0000FE 6010               bra.s   Display            and Display the message. -----------------+|
                    *                                Progress report.  Get message ptr,        ||
 000100 41FA FF9C   Report lea      ProgMsg(pc),a0    <----------------------------------------|+
 000104 200D               move.l  a5,d0              load the checked address,                |
 000106 6100 FF08          bsr     hexadecimalize     make it hex for Display. <-->            |
 00010A 5188               sub.l   #8,a0             Regain pointer to the message,            |
 00010C 7609               move.l  #P_SIZ,d3          get the size for the write,              |
 00010E 4855               pea      (a5)              push a return ptr to the new Worm,       |
                    *                                 and drop through into Display.           v

                    ****  Display   ***************************************************************
                    * Display is an implementation-dependent scheme for reporting the Worm's
                    * progress.  Upon entry, A0 contains a pointer to a string to Display, and D3
                    * contains the length of the string to Display.
                    *
                    *     Entry:    d3 -- number of bytes to display.
                    *               a0 -- address of a string to display.
                    *
                    *     Uses:     d0 -- file descriptor of stdout.
                    *               a1 -- scratch register for pointing to SysCall param block.
                    *
                    *     Stack:    as needed by system call.
                    *
                    ********    B E G I N   S Y S T E M - D E P E N D E N T   C O D E   ********
 000110 2F03        Display move.l  d3,-(a7)     Load the byte count, <---------------------+
 000112 2F08                move.l  a0,-(a7)       the actual string pointer,
 000114 3F3C 000D           move.w  #write,-(a7)   and the system call index,
 000118 204F                move.l  a7,a0          point to the syscall parameter block,
 00011A 7001                move.l  #1,d0          load file descriptor for stdout,
X00011C 0000 0001           SYS     indx           and write the message. <-->
 000120 DFFC 0000 000A      add.l   #10,a7       Remove the params from the stack, and
 000126 4E75                rts                    return somewhere. -->
                    *
                    * For lack of a better place to put it, the system- dependent exit code is here.
                    *
X000128 0000 0005   Exit    SYS     term         Terminate this program.  (System dependent.)
                    ********    E N D   S Y S T E M - D E P E N D E N T   C O D E   ********

                    ****  Disable, Enable   ***************************************************
                    * These routines provide the exclusion mechanism for the non-interruptible code
                    * in Worm at Crawl.  These routines must execute in supervisor state, therefore
                    * they are executed via the TRAP exception instruction.  Enable requires that
                    * D1 be preserved from the preceding Disable.
                    *
                    *     Uses:     SR -- interrupt mask is raised and lowered.
                    *               d2 -- scratch register for restoring original interrupt mask.
                    *               d1 -- scratch register storage place for old interrupt mask.
                    *
                    ********    B E G I N   S Y S T E M - D E P E N D E N T   C O D E   ********
 00012C 40F9 0000 000A Disable move    sr,10        Grab the status register,
 000132 0241 0300              and.w   #$0300,d1      keep only the interrupt bits,
 000136 027C 0300              and     #$0300,sr      and disable all interrupts
+00013A 0000 0008 0000         SYS     cpint,SIGTRAP2,Disable <-->
 000146 4E77                   rtr                    before entering critical code region. -->

 000148 40C2           Enable  move    sr,d2        Regain the status register,
 00014A 8441                   or.w    d1,d2          reset the previous interrupt level,
 00014C 46C2                   move    d2,sr          and enable the proper interrupts
+00014E 0000 0008 0000         SYS     cpint,SIGTRAP3,Enable <-->
 00015A 4E77                   rtr                    before exiting critical code region. -->
                       ********    E N D   S Y S T E M - D E P E N D E N T   C O D E   ********

                       ****  Worm     ***********************************************************
                       * Worm is a self-modifying, self-relocating procedure which starts at some
                       * location in high memory and works its way down to its end address,
                       * periodically reporting its progress.
                       *
                       * The loop at Crawl depends strongly on the 68000 prefetch mechanism.  This
                       * loop will not work on a 68020 machine (which has a 64 entry cache), nor on
                       * most simulators (which often do not bother to simulate prefetch accurately).
                       * This loop will also not work with the TRACE bit set, and must be protected
                       * from all interrupts, including page faults in virtual memory systems.
                       *
                       * When this loop moves the DBNE long word at Crawl+4, it overlays the MOVE.L
                       * and the CMPM.L at Crawl.  The CMPM.L is in the prefetch queue, so it gets
                       * executed even though its memory image has just been clobbered.  The DBNE is
                       * fetched, and its execution flushes the prefetch queue as is the case with all
                       * branches.  Execution continues with the copy of the DBNE just moved, which
                       * executes again, branching to Crawl-4, the new loop location.  Note that the
```

```
                                 * loop count gets decremented twice in this scenario, removing the need for the
                                 * usual predecrement before entering the loop.
                                 *
                                 *
                                 *      Entry:    d7 -- length of Worm in long words.
                                 *                d6 -- base of memory area to test.
                                 *                d5 -- address mask for display boundary.
                                 *                a5 -- first long word address of Worm at present.
                                 *                a4 -- first long word address of Worm's original image.
                                 *                a3 -- display manager's address.
                                 *
                                 *      Exit:     d0 -- W_LONGS complement of pass count if error.
                                 *                a5 -- entry value less relocation, i.e.: next pass entry value.
                                 *                a1 -- address pass/fail report value.
                                 *
                                 *      Uses:     d0 -- decrementing Worm length.
                                 *                a2 -- incrementing COMPARE address.
                                 *                a1 -- incrementing TO address.
                                 *                a0 -- incrementing FROM address.
                                 *
                                 *      Unused:   d4, d3, a7, a6.
                                 *
00015C 3007                      Worm     move.w   d7,d0          Restore the Worm's length,
00015E 204D                               move.l   a5,a0              its starting point,
000160 244C                               move.l   a4,a2              and its original address.
000162 43ED FFFC                          lea      -4(a5),a1      Get the destination for this pass.
                                 ********    B E G I N   S Y S T E M - D E P E N D E N T   C O D E   ********
000166 4E42                               trap     #DISABLE       Don't interrupt this critical passage! <-->
                                 ********    E N D   S Y S T E M - D E P E N D E N T   C O D E   ********
000168 2298                      Crawl    move.l   (a0)+,(a1)     Move a long word piece of Worm, <--------+
00016A B589                               cmp.l    (a1)+,(a2)+        and check it against the original,   |
00016C 56C8 FFFA                          dbne     d0,Crawl           one long word at a time. -------------+
                                 ********    B E G I N   S Y S T E M - D E P E N D E N T   C O D E   ********
000170 4E43                               trap     #ENABLE        Allow interrupts -- critical section over. <-->
                                 ********    E N D   S Y S T E M - D E P E N D E N T   C O D E   ********
000172 598D                               sub.l    #REL_SIZ,a5    Update the new Worm address,
000174 4E71                               nop                         keep the whole thing on long boundary,
000176 4ED3                               jmp      (a3)               report to the Manager. -->


                                 *
                                 * The following pattern (which is notoriously hard on 16-bit dynamic RAM
                                 * memories) gets left in memory and can be checked later if desired.
                                 *
```

(continued on next page)

Listing One (Listing continued, text begins on page 114.)

```
000178                  Droppings
000178 5555 AAAA                 DC.L     $5555AAAA      Pattern to be left in RAM.
       0000 0020        W_SIZ    EQU      *-Worm         Length of self-relocating code, in bytes
       0000 0008        W_LONGS  EQU      W_SIZ/4             and longs.

                        ****  Init      **************************************************************
                        * Init performs system-dependent initialization and sets up registers for use
                        * of Worm and Manager.  Init then copies the Worm into the top of test memory
                        * and starts the Worm crawling.
                        *
                        *     Entry:    not applicable.
                        *
                        *     Exit:     a5 -- Worm's test image address at top of memory to be tested.
                        *               a4 -- Worm's permanent image address.
                        *               a3 -- Manager routine pointer.
                        *               d7 -- length of Worm in long words.
                        *               d6 -- base of memory area to test.
                        *               d5 -- address mask for testing display boundary.
                        *
       0000 017C        Ovrly    EQU      *              This area will be overlaid with the worm.
00017C 576F 726D 206D LogMsg     DC.B     'Worm memory tester, '
000190 2448 6561 6465            DC.B     '$Header: worm.a-v 1.2 86/03/24 01:44:36 jans Exp $'
0001C4 0D4D 656D 6F72            DC.B     CR,'Memory checked down to location:',CR
       0000 006A        L_SIZ    EQU      *-LogMsg

0001E6 00                        EVEN
                                 GLOBAL   Init
0001E6                  Init
                        *
                        * First, perform some system-dependent initialization: set up the TRAPs needed
                        * to protect the Worm from interrupts, protect the area to be tested from page
                        * faults, and write a welcome message.
                        *
                        ********   B E G I N   S Y S T E M - D E P E N D E N T   C O D E   ********
+0001E6 0000 0008 0000           SYS      cpint,SIGTRAP2,Disable  Set up the exception handlers for the
+0001F2 0000 0008 0000           SYS      cpint,SIGTRAP3,Enable     interrupt exclusion routines.
+0001FE 0000 0039 0000           SYS      memman,1,MemBeg,MemEnd  Protect memory image from page faults.
 00020E 7001                     move.l   #1,d0                   Prepare and write a stdout
+000210 0000 000D 0000           SYS      write,LogMsg,L_SIZ        welcome message.
                        ********   E N D   S Y S T E M - D E P E N D E N T   C O D E   ********
                        *
                        * Next, set up registers that will be used by the Worm and Manager.
                        *
00021C 2A3C 0000 03FC            move.l   #D_MASK,d5     Get the Display address boundary mask.
000222 41FA FF58                 lea      Ovrly(pc),a0   Load the lowest address to test
000226 2C08                      move.l   a0,d6            into a data register for comparison,
000228 47FA FE7E                 lea      Manager(pc),a3   get the Display Manager's address,
00022C 49FA FF2E                 lea      Worm(pc),a4      the Worm's non-crawling image address,
+000230 2A7C 0000 7FE0           move.l   #MemEnd-W_SIZ,a5 and the high-mem Worm start address.
000236 3E3C 0008                 move.w   #W_LONGS,d7    Get the Worm's length in longs.
                        *
                        * Finally, move the Worm to the top of memory to be tested.
                        *
00023A 204C                      move.l   a4,a0          Get a copy of Worm's permanent image pointer,
00023C 224D                      move.l   a5,a1            its test image pointer,
00023E 3007                      move.w   d7,d0            and its length in longs.
000240 5340                      sub.w    #1,d0
000242 2290             MoveWorm move.l   (a0),(a1)      Move, and compare <-------------+
000244 B388                      cmp.l    (a0)+,(a1)+      a long word of the Worm        |
000246 56C8 FFFA                 dbne     d0,MoveWorm      at a time. -------------------+

00024A 4A40                      tst.w    d0             Exit loop by error, or countdown?
00024C 6A00 FE5A                 bpl      Manager          Error, go Report it. -->
000250 4ED5                      jmp      (a5)             Countdown.  Start Crawling! -->
       0000 0252        C_SIZ    EQU      *-MemBeg       (Size of non-relocating code.)

000252 0000 0000 0000            DS.B     MEM_SIZ-C_SIZ
       0000 8000        MemEnd   EQU      *
                                 ENDDEF
       0000 01E6                 END      Init           (Set transfer address to the Init.)

0 Errors detected.

SEGMENT SIZES
TEXT SEGMENT = 000000
DATA SEGMENT = 008000
BSS SEGMENT  = 000000
```

End Listing

16-BIT SOFTWARE TOOLBOX

MS-DOS Books

In the May 1986 16-Bit Toolbox column, I briefly reviewed some books on MS-DOS assembly-language programming. Since I wrote that column, another interesting book has appeared:

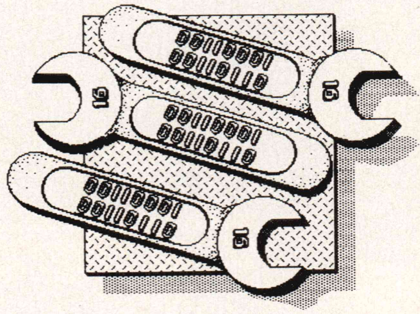
Angermeyer, John, and Jaeger, Kevin. *MS-DOS Developer's Guide.* Indianapolis, Ind.: The Waite Group/Howard K. Sams and Co., 1986. 440 pages with index. $24.95.

Compared to the previous published efforts on this topic, this is a remarkable book and is the first book on MS-DOS programming that I would actually characterize as being directed at advanced assembly-language programmers (that is, typical *DDJ* readers). Topics covered that have been neglected or ignored in nearly every other book published to date include detailed instructions on use of the advanced features of MASM (macros and conditional assembly), design and coding of memory-resident utilities and run-time libraries, memory management, installable device drivers, local-area networks, real-time programming under MS-DOS, disk-layout and file-recovery information, and the functional differences beteen MS-DOS versions.

The text of the book is well supplemented with assembly-language examples in the form of subroutines or complete working programs. The authors have included many tidbits of information and programming pearls (such as the method for removing a memory-resident program) that are obviously derived from extensive personal experience. I predict that nearly every reader of this column will find something new and useful

by Ray Duncan

in this book and that they will consider it money well invested.



The BIOS Done It

The short description of my problems with the PC/AT VDISK program in the April 1986 column drew a flurry of mail from readers. The first, and most caustic, reply came from George Scotten of Springfield, Vermont. Mr. Scotten wrote: "Ray Duncan's column is a classic case of RYFM (read your fact-filled manual)! . . . Although he claims that he and his coworkers spent a lot of time poring over the IBM tech ref manual, the time might have been better spent reading it. . . . Interrupts *0f1–0ffh* are listed as reserved interrupts, and anyone using a reserved interrupt deserves what they get. . . . It only took this amateur 30 seconds to resolve his problem. . . ."

Well, this letter from Mr. Scotten rattled me for a few minutes, I must admit. I leaped out of my chair and consulted my PC/XT and PC/AT technical reference manuals once again. No, I wasn't hallucinating: although the manuals clearly state that some interrupt ranges are "Reserved" (for example, *28h–3fh*, *40h–5fh*, and *80h–85h*), the interrupts *0f1h–0ffh* are definitely tagged "Not Used" rather than "Reserved." (See *PC/AT Technical Reference Manual*, p. 5-6, and *PC Technical Reference 2.02 Manual*, p. 2-8.)

A considerably more helpful letter came from Thomas Thurston, of Intel Corp., who wrote: ". . . Actually, the problem is not in VDISK at all but in the PC/AT ROM BIOS function that VDISK uses to access extended memory (BIOS interrupt *15h*, function *87h*, pp. 5-150 to 5-155 of the *PC/AT Technical Reference Manual*). This BIOS function creates 80286 protected mode descriptor tables and then switches to protected mode so that it can access extended memory directly. As you noted in your column, to get out of protected mode, the BIOS sets a special value in the CMOS RAM, outputs a signal to cause a *RESET*, and then halts. In the power-up sequence after the *RESET* signal is received (pp. 5-33 to 5-35), the value from the CMOS RAM is checked. If it indicates that the *RESET* was caused by a shutdown, control is returned to the code that requested the shutdown.

"There are some side effects of going through the *RESET* sequence. The registers do not have the values they had before the shutdown. In particular, the stack registers (*SS* and *SP*) are lost. *RESET* initializes *SS* to 0 and leaves the value of *SP* undefined. The BIOS code recognizes this, but it handles it in a way that causes problems. After the power-up code recognizes that a shutdown has occurred, before transfering control back to the point that requested the shutdown, it initializes *SS* to *0030h* and *SP* to *0100h* (p. 5-34 and p. 5-29). This area of memory (absolute addresses *300h–400h*) overlaps the end of the interrupt vector table. During system initialization, this isn't a problem. However, when the area is used as a stack during the *RESET* sequence to come out of protected mode, some of the entries in the interrupt vector table are trashed.

"When control is returned to the point in the BIOS code that requested the shutdown (p. 5-152), one of the first things it does is restore the user's stack (the values of *SS* and *SP* were saved previously). Before restoring *SS* and *SP*, however, the code actually does two procedure calls (which will cause two return addresses to be pushed on the stack). One of the procedures calls another procedure, which uses the stack to save the value of *CX*. In all, three words (6 bytes) of

stack space are used before the user stack registers are restored.

"With the Intel 8086 architecture, *SP* is decremented before pushing values onto the stack. Thus the last 6 bytes of the interrupt vector table are used as stack space and destroyed by this BIOS function. Each entry in the interrupt vector table uses 4 bytes. This means that the vectors for interrupts *0feh* and *0ffh* are always lost (after a transition to protected mode and back again).

"It seems to me, however, that the problem is even more severe than this. When *SS* and *SP* are initialized by the *RESET* sequence (p. 5-34), the interrupts are turned off before and on again afterward. The reason for turning the interrupts off is to avoid the problem of an interrupt occurring while the stack is in an undefined state (after setting *SS* but before setting *SP*). However, it is not necessary to turn the interrupts off if *SP* is loaded during the very next instruction after loading *SS* because the 286 always inhibits interrupts until completing the next instruction after loading *SS*.

"In fact, turning the interrupts off and then on again causes problems because it leaves the interrupts on later, while the subsequent code (pp. 5-152 and 5-153) assumes that interrupts are off. In the first place, more than just the last entries in the interrupt vector table will be trashed if any interrupts occur before the user's stack is restored. Second, the code that restores the user's stack does not turn interrupts off when restoring *SS* and *SP*, and it executes an additional instruction after loading *SS* before loading *SP*. An interrupt at this point would trash arbitrary locations in the user's stack segment." *[These might overlap and destroy locations in the user's code and data segments.—Ray]*

Hans Pufal, Tom Roberts, and Bob Sharpe, among others, also sent detailed explanations of the VDISK problem giving essentially the same information. Hans Pufal threw in a little conundrum for the amusement of *DDJ* readers (Table 1, below), and Bob Sharpe also added: "There is a block of interrupts specifically reserved for user programs (interrupts *60h*–*67h*). The only 'problem' with using these interrupts is one of conflicting usage with other programs (for example, the Expanded Memory Manager for the Intel Above Board and other Lotus/Intel/Microsoft EMS implementations uses interrupt *67h*). This need not be a problem for any application that can save the old interrupt vector, use the interrupt during execution, and finally restore the original interrupt.

"It might be noted that the original PC/AT BIOS has quite a collection of errors (for example, see the clever way the zero flag is set only a few lines later on p. 5-153 and followed by *IRET*). The newer 'infamous' BIOS that cripples the system to a 6-MHz clock rate includes fixes for nearly all the BIOS problems we had located."

Microsoft Macro Assembler

In my May 1986 column, I noted a new problem that appeared in Version 4 of the Microsoft Macro Assembler such that end-of-file marks (*1ah*) don't seem to be recognized at the end of *include* files, resulting in confusing error messages if the text file was written with certain editors (such as WordStar in nondocument mode). The technical-support people at Microsoft have supplied a patch that will correct this problem (see Listing One, page 96).

With regard to another potential problem, David Gwillim of Los Angeles wrote: "If you are getting strange errors from your MASM, or strange results or even crashes from your assembled and linked programs, there is an insidious bug that may be responsible.

"Versions of MASM other than the original IBM MASM 1.0 all expect to see a *CR* and an *LF* at the end of each line

```
and   al,0fh
add   al,90h
daa
add   al,40h
daa
```

Table 1: *Pop quiz from Hans Pufal: What does this code do?*

16-BIT
(continued from page 109)

before they will recognize line termination. MASM 1.0 will work with either.

"The nonacceptance of a lone *CR* at the end of a line where there is a comment field (which is most lines in an ASM file) causes the next line simply to become part of the comment field of the previous line, effectively removing it from the assembler's view!

"In the case where no *LF*s are used at all, this will be immediately obvious, but if there are just a few *CR*-only lines, then you will have many strange occurrences—symbol-not-defined errors, crashes running a program that looks just fine in the source text, and so on.

"Because many text editors don't seem to care whether there is an *LF* accompanying each *CR*, the occasional omission of an *LF* can be hard to find. One simple way to locate these is to do a *COPY FILENAME.ASM PRN*, and the printer will print the lines that don't have an *LF* separating them on top of each other."

DOS File Handles

The discussions of the 20-handle limit in the December 1985 and May 1986 16-Bit Toolbox columns generated a great deal of interest and discussion among *DDJ* readers. A particularly unique work-around was contributed by Paul Adams of Shelbyville, Kentucky, who wrote: "The letter from Dan Daetwyler quoted in your December 1985 column was the first I had heard of MS-DOS' limit of 20 file handles per process. This came as an unpleasant shock to me because, like Dan, I was planning a database application that would certainly require more than 20 open files. Although, Dan's letter left the impression that this restriction is new with Version 3 of DOS, I have found that it applies to DOS 2.0 as well.

"There is a way around. The clue was provided in the January 1986 *PC Tech Journal*. A Tech Notebook by Stan Mitchell describes the mechanism DOS uses to redirect file handles.

"A program segment prefix (PSP) contains a table of 20 bytes starting at offset *18h*. When a file (or device) is opened, the handle returned by DOS is an offset into this table. The byte at offset *18h+handle* in the PSP will contain what I call the real handle. The real handle represents an entry in an internal DOS table. The default size of this table allows for eight real handles. This can be changed with the *FILES* command in CONFIG.SYS. If *FILES = 255* is included in CONFIG.SYS, the real handle has a range of values from *0* to *feh*. A real handle of *ffh* always means the file is closed.

"The first three real handles are predefined by DOS as:

0—aux device
1—console
2—printer

"The result of this redirection is to allow child processes to inherit the open files of the parent process. The only use DOS seems able to make of this is to redirect standard input and output.

"By the way, as far as I can tell, a process is defined by a PSP. The currently active PSP can be determined by DOS function *62h*. The only way I know to change the active PSP is to create a child process using DOS function *4bh*. *[See also Ross Nelson's explanation of MS-DOS process IDs in the May 86 column.—Ray]*

"The way to open more than 20 files from a single process is to trick DOS into reusing one of the table entries in the PSP. I call this technique handle packing.

"To open or create a file using handle packing:

1. Open or create a file with the appropriate DOS handle function.
2. The *dos_handle* is the handle returned by DOS. The *real_handle* is the byte at offset *18h+dos_handle* in the PSP. Save the real handle for use when performing I/O on the file.
3. Replace the byte at offset *18h+dos_handle* with *ffh* so the *dos_handle* can be reused.

"For all other file functions using handle packing:

1. Save the *real_handle* found at offset *18h+19* so it can be restored later. (*18h+19*, the last handle in the PSP table, was arbitrarily selected.)
2. Place the *real_handle* of the desired file at offset *18h+19*.
3. Move *19* to the *bx* register and exe-

16-BIT
(continued from page 110)

cute the desired DOS function.
4. Restore the *real_handle* that was saved in step 1.

"The functions in HPKIO.ASM [Listing Two, page 96] provide the basic handle-packing I/O system for use with the small-memory model of Version 2.1 of the Computer Innovations C86 compiler. The program HTEST.C [Listing Three, page 101] is used to demonstrate the functions. To try HTEST, create an empty *\TEST* directory. The number of files created by HTEST will be three less than the number of files specified in CONFIG.SYS.

"This solution is obviously something of a hack because the redirection scheme is not documented and thus may change in future releases of DOS.

"Considering current developments in mass-storage devices, the IBM PC family of machines could be used for some sizable applications. This arbitrary limit of 20 files, however, disqualifies these machines (under MS-DOS, at least) for serious database applications. I am surprised that there have not been more complaints. Is I/O redirection really worth cutting the file limit from 255 to 20?"

Resident Programs and File I/O

For those *DDJ* readers writing the next SideKick or Ready, Gary Cramblitt has got a question for you: "How can a resident MS-DOS program perform disk file I/O without trashing an existing program also doing disk I/O? An on-line notepad program, for example, allows the user to press a special key at any time, even while running some other program. A window opens up on the screen, the user enters his or her note, and the note is recorded in a notepad file on the disk.

"The problem is that MS-DOS is not reentrant. When the user presses a special key, the processor may be currently executing inside an MS-DOS file I/O routine. If the on-line notepad program calls MS-DOS disk I/O functions, the original program's disk I/O will get trashed or the computer will hang.

"I've come up with several ideas on how to solve this problem, but so far, none of these solutions is ideal. One technique can be used if the target computer has a timer hardware interrupt. The MS-DOS Get Critical Flag Address function (interrupt *21h*, function *52h*) can be used to tell when the processor is not executing code inside MS-DOS. A timer interrupt routine could keep checking this flag. When DOS is no longer critical (and therefore is no longer inside an MS-DOS disk I/O routine), then the notepad program can safely request its disk I/O.

"There are three problems with this technique. First, it is hardware-dependent because you must have knowledge of (and capture) the particular computer's hardware timer interrupt. Second, the Get Critical Flag Address function is available only in MS-DOS, Version 2, and later. *[Note: This function is not documented for PC-DOS systems, although it seems to work.—Ray]* Third, a lot can happen between the time the user presses the special notepad activating key and the time MS-DOS is not in a critical section. For some applications, this won't be a big problem. However, I have in mind a program to dump the graphics screen of my computer (a Z-100) to a disk file. Too much can happen on the screen between the time the key to request the screen copy to disk is pressed and the time MS-DOS becomes noncritical.

"Another technique for solving the reentrancy problem would be to save the notes in memory—like a special RAM disk. The user must run a special program before shutting off his or her computer to copy the notes from the memory file to floppy disk. This solution suffers from two obvious deficiencies. First, it requires more memory. Second, if users forget to run the special program or if they lose power, their notes are lost forever.

"A third technique would be for the on-line notepad program to do its own file handling, reading and writing physical disk sectors directly. I think the problem with this approach is obvious." **DDJ**

(Listings begin on page 96.)

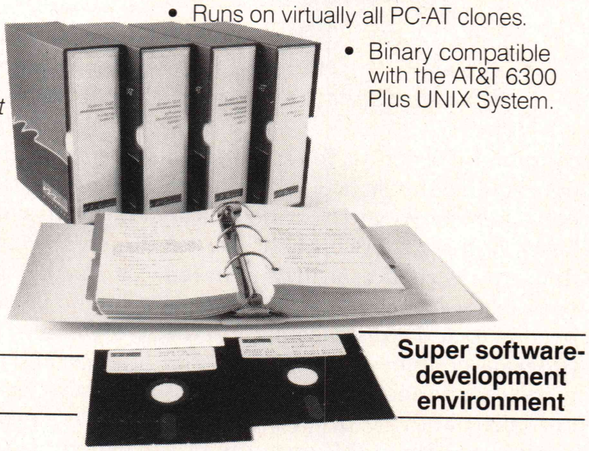
Vote for your favorite feature/article.
Circle Reader Service **No. 6.**

If you think you can't afford a UNIX® system. we've got a $160 surprise.

Turn your PC into a multi-user system.

Convert your IBM PC-AT (or compatible) into a multi-user/tasking UNIX work station—at *absolutely the best price* anywhere, any time. Based on the AT&T-certified UNIX System V/286, the MICROPORT SYSTEM V/AT is designed for use in virtually any computer environment, from office automation to software development.

- Dynamic disk buffer allocation provides RAM disk performance for systems with large memory configuration.
- Runs on virtually all PC-AT clones.
- Binary compatible with the AT&T 6300 Plus UNIX System.



Over 200 utilities come standard.

Grep, awk, sort, split, cut, paste, vi and ed (and many more) now let you search and modify files, make use of electronic mail, emulate terminals, calculate electronically, convert data and publish.

SYSTEM V/AT is more than a look-alike. It was derived from AT&T's own UNIX System V release 2 iAPX286. It thereby contains standard System V features the competitors don't support, such as the powerful symbolic debugger, sdb, the shell-layering job-control facility and the F77 Fortran compiler, as well as programming tools such as ctrace, cflow, and bs. Also standard is File System Hardening which greatly reduces data loss in a power failure.

Super software-development environment

We've provided everything: Make, yacc, lex, sccs, cflow, ctrace plus every standard System V software-development tool. The F77 Fortran compiler. And the AT&T Portable C compiler for the 286. Both C and Fortran compilers generate 287 instructions directly—for systems not containing 287 math coprocessors, a kernel-resident IEEE-compatible 287 emulator is provided. The large-model code produced by the compiler is among the densest and fastest currently available.

So, how do we do it?

MICROPORT offers SYSTEM V/AT at a fraction of the price of the competitors simply because we build on the generic System V/286 product from AT&T. This entire utility package from the certified release has been copied directly to SYSTEM V/AT—without so much as a recompile. Not only does this mean that MICROPORT can offer SYSTEM V/AT at a remarkable low price, it also *guarantees a level of quality* present in few (if any) other UNIX-system implementations. (And, since our staff was part of the group that implemented the standard System V/286 port for Intel, MICROPORT can offer comprehensive support for the system, as well.)

And a dollar change

The price is even better than you thought. Order right away and we'll return one silver dollar just as rapidly, with your product shipment. (If you'd like a little more time we'll apply that dollar to the cost of a brochure—which we'll send right away too.)

NOTE: Educational institutions—Call about our nominally-priced site licensing and source code

60 DAY MONEY-BACK GUARANTEE

Want some more features?

- Console driver providing ANSI terminal interface for monochrome, CGA, Hercules and EGA cards.
- Multiple Virtual consoles allow up to four virtual windows of operation.
- Record and File Locking
- Supports the 286's 16 megabyte virtual address space and fully utilizes its other advance features.
- Supports all standard IBM drive types and most non-standard hard-disk drives.
- Requires only one hard-disk partition, and allows DOS to reside on the same hard disk.
- Provides utilities to transfer files to-and-from DOS file systems.

And a dollar change.

To order: Complete the information below. Your attractively-packaged and fully-documented order will be shipped within two weeks.

MICROPORT SYSTEMS, INC.
4200 Scotts Valley Drive
Scotts Valley, CA 95066
408/438-UNIX or 800/PC2-UNIX (outside CA)

SYSTEM V/AT

☐ **RUNTIME SYSTEM** Includes the SYSTEM V/AT operation system and over 200 utilities, for two users.
QUANT. ________ **$160.00**

☐ **SOFTWARE DEVELOPMENT SYSTEM** The complete Software Generation System for 286 development.
QUANT. ________ **$169.00**

☐ **TEXT PREPARATION SYSTEM** Includes nroff, troff, spell and other programs.
QUANT. ________ **$169.00**

☐ **THE COMPLETE SYSTEM** Contains all three packages indicated above.
QUANT. ________ **$439.00**

☐ **OPTIONAL** three to eight-user upgrade.
QUANT. ________ **$99.00**

Subtotal: ______

(CA residents add 6.5% tax per copy): ______

Shipping and handling charges (In the USA, $14.00; in Canada, $18.00; and in Europe, $110 . . . . : ______

TOTAL DUE: ______

NAME ____________

TELEPHONE ____________

ADDRESS ____________

CITY ____________

STATE ________ ZIP ________

COUNTRY ____________

☐ VISA ☐ MASTERCARD ☐ BANK DRAFT ☐ CHECK

CARD NUMBER ____________ EXP DATE ______

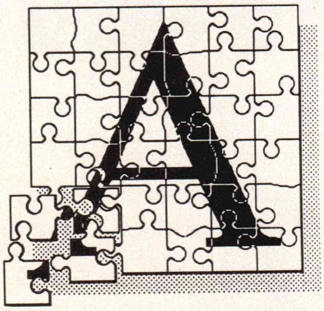
☐ Send a brochure only and keep me on your mailing list, please.

M I C R O P O R T

UNIX and DWB are trademarks of AT&T
IBM and IBM PC-AT are trademarks of IBM CORPORATION.
SYSTEM V/AT is a trademark of MICROPORT SYSTEMS, INC.

Circle **no. 154** on reader service card.

The Worm Memory Test



No, this is not a method for quantifying the mental retentive powers of certain long, cylindrical invertebrates. It is a test that could help to diagnose certain types of computer memory errors. The Worm memory test (see Listing One, page 102) uses a dynamically executing program as the actual test data. Unlike previous memory test programs of this type, this worm has a special twist—it is able to overlay itself while it is executing, thanks to the MC68000's prefetch register.

Some Fetching Facts

Never heard of the prefetch register? To understand how the memory test works, it might help to review the way the MC68000 fetches and executes instructions. The MC68000 uses instruction pipelining in order to speed execution. There is, in effect, a 16-bit register between the data bus and the instruction decoding logic. When an instruction is executed, the opcode for that instruction is first loaded into the prefetch register (often while the previously fetched instruction is being executed), then the instruction is moved into the instruction decoding register, where it is executed. The net effect is that the processor usually has a handle on the next thing it is supposed to do.

Prefetch works fine most of the time, but it does slow things down during certain operations. If the instruction being executed causes a nonsequential instruction to be executed, execution may be either faster or slower. In the case of a conditional

by Jan W. Steinman

branch instruction, a branch taken is quite fast because the prefetch register already holds the displacement that must be added to the program counter in order to fetch the next nonsequential instruction. A branch not taken, however, will be a little faster if it is a short branch because the next instruction is already in the prefetch register and the two clocks needed to add a displacement to the program counter can be saved. The worst case happens when a branch is not taken and the branch displacement is 16 bits. In this case, the processor has useless information in the prefetch register and must flush that information before it can fetch the next instruction.

Other nonsequential instructions cause an immediate flush of the prefetch register and use an extra four clocks simply to restart the pipeline. One exception is the decrement-and-branch instruction, which like the taken branches benefits from having the branch displacement handy. (The MC68010, with its 32-bit prefetch register, actually executes many 16-bit instructions out of the prefetch register if they precede a decrement-and-branch instruction.)

How the Worm Crawls

Worm depends on these characteristics of pipelining in order to overlay itself while it is running, but it needs some management and control in order to be useful—a *Worm* on the loose would quickly destroy all memory! Besides *Worm*, a complete memory test requires two additional parts: an initialization sequence and a routine for controlling *Worm* and reporting its findings.

The initialization routine, *Init*, has some special characteristics and includes most of the system dependencies. It is executed only once—at the beginning—and is therefore throw-away code. This is why it is placed last—*Worm* actually crawls right over its initialization code in this implementation. The registers are set up to the specifications of *Worm*, and several important system functions are performed. In particular, it is important that page faulting does not occur in systems that support virtual memory, and if special hocus-pocus is needed to turn off interrupts, it should be done here.

Manager exercises control over *Worm* and is responsible for communicating errors it discovers and displaying progress messages if desired. When *Manager* is entered upon completion of a *Worm* pass, it must decide if it has been entered because of an error or simply as a point of control. If there has been an error, *Worm* is no longer runnable, so *Manager* will have to report the error and terminate. If no error is detected, *Manager* must check the progress of *Worm* to keep it from consuming all memory. At this point, *Manager* can decide enough memory has been checked to warrant a progress report of some kind.

The real heart of the whole thing is, after all, *Worm*. *Worm* simply replicates itself, one long word lower in memory, while comparing the new copy of itself against the original, which never executes. *Worm* may be the heart of the memory test, but the three instructions starting at *Crawl* are where the magic happens. This loop starts at the beginning of *Worm* and copies the first long word down to *Worm−4*. It continues with each additional long word, until it gets to the long word at *Crawl+4*, which is a *dbne* instruction with its 16-bit displacement. The preceding *move.l* and *cmp.l* have already been copied down.

At this point, it becomes a little difficult to keep track of what is data and what is code. When the *move.l* is in the instruction decode register,

Jan W. Steinman, 2002 Parkside Ct., West Linn, OR 97068

ready to be executed, the following *cmp.l* is in the prefetch register, waiting its turn to be executed. When the *move.l* at *Crawl* executes, it moves the *dbne* instruction into the location it and the following *cmp.l* are currently occupying. The processor has no way of knowing it has just invalidated its prefetch register, so it continues—moving the *cmp.l* instruction into the instruction decode register and moving the following *dbne* into the prefetch register. The *cmp.l* executes, comparing the *dbne* just moved against the original while moving the branch displacement for the *dbne* into the prefetch register.

Assuming the compare was successful, the *dbne* executes, decrementing *d0* and branching backward 4 bytes to where the *move.l* used to be. The prefetch register is flushed because of the branch, so the value at that location is loaded into the prefetch register and immediately into the instruction decode register. But what is loaded? A copy of the *dbne*, complete with the same negative displacement value. The condition codes have not changed, and the count register *d0* should not be anywhere near 0, so the copy of the *dbne* gets executed identically to its predecessor, which still resides in the next long word. The *dbne* copy branches to the *move.l* copy, and the loop continues moving the code down 4 bytes. (See Table 1, above.)

When the count register *d0* underflows, the *dbne* copy drops through, interrupts are enabled, *Worm*'s dynamic image pointer *a5* is adjusted to point to the new *Worm* copy, and *Worm* reports back to *Manager*. Note that none of the *Worm* code is ever executed before it has been compared and verified.

It is vitally important to disable interrupts when the *move.l* overlays itself and the following *cmp.l*. An interrupt at this point causes the prefetch to be flushed when the interrupt is serviced. Upon return from the interrupt, the displacement part of the *dbne* (hex fffa) will be fetched as an instruction. This will cause a "line 1111 emulator exception" unless your system has a coprocessor with an ID code of 7, but either way *Worm* will be broken and the memory test will fail. And of course, it is important that the length of *Worm* remains a multiple of 4 if you decide to modify it!

But What Good Is It?

I originally developed the MC68000 Worm test for an embedded processor application that was having dynamic-RAM refresh problems. It was discovered that conventional RAM tests, which move smoothly up through consecutive addresses, were masking the problem by unintention-

	Before		After	
Crawl−4	. . .		move.l	(a0)+,(a1)
Crawl−2	. . .		cmp.l	(a1)+,(a2)+
Crawl	move.l	(a0)+,(a1)	dbne	d0,−6 <--+
Crawl+2	cmp.l	(a1)+,(a2)+		¦
Crawl+4	dbne	d0,−6 ----------------------------------+		

Table 1: *The test in action*

Circle **no. 148** on reader service card.

Circle **no. 152** on reader service card.

RIGHT TO ASSEMBLE
(continued from page 115)

ally providing software refresh. The test is not long enough to cause a complete cycle of all a dynamic RAM's row-address-strobe (RAS) lines and was able to help diagnose the problem.

In the form presented, this implementation is useful primarily as an illustrative example of position-independent coding, modular design, and, of course, a unique use of the prefetch register. It could be put to practical use in several ways.

The best use of the memory test might be to have it running continuously, as a very-low-priority task. *Manager* would have to take some of the responsibility of *Init* by allocating test memory and restarting *Worm* when it has finished testing a buffer. The interrupt disabling code may be simpler on systems without virtual memory—on the Amiga it is a simple memory store.

Virtual-memory systems would also need to add code to branch around the interrupt disabling code on the copy of the first long word only, which would allow the memory test to generate page faults whenever it first crosses a page boundary. To make it practical in such systems, *Manager* would have to access the memory-management hardware in order to map faulty virtual locations to broken chips.

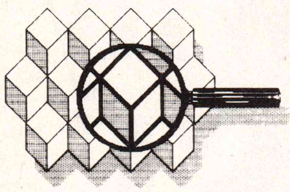
The *Worm* routine itself can hold much more code if desired. I originally had much of *Manager*'s decision code in *Worm*, which did speed it up but at the expense of simplicity. In a message-based system, such as the Amiga, *Manager* could be totally deleted. *Worm* could contain all the task code, merrily crawling through any available RAM it could find and sending error reports through intertask messages—all with minimal impact on the user.

DDJ

(Listing begins on page 102.)

Vote for your favorite feature/article.
Circle Reader Service **No. 7**.

OF INTEREST



As artificial intelligence moves more into the real world and out of the laboratory, we are seeing more and more tools designed to help application developers use AI techniques in their programs. In particular, PROLOG and expert systems tools seem to be getting hot. Where it will lead is anybody's guess—we are still far from true "common sense" AI programs, and there are few products claiming to use AI that are much more than sophisticated database tools. What new developments are on the horizon? Will true AI finally become more than a complex toy? What about the poorly understood sister fields, pattern recognition and natural-language interfaces?

IntelligenceWare has announced a new expert systems tool, Experteach-II, a comprehensive guide including complete LISP and PROLOG interpreters for the IBM PC. The product includes on-line and written tutorials, plus tools and source code for expert systems based in LISP, PROLOG, dBASE II, or Pascal. It's priced at $475. Reader Service No. 16.
IntelligenceWare Inc.
9800 S. Sepulveda Blvd.
Ste. 730
Los Angeles, CA 90045
(213) 417-8896

PML Systems announces BEAGLE (Bionic Evolutionary Algorithm Generating Logical Expressions), a data analysis system that uses artificial intelligence to construct rules and inferences from a knowledge database. BEAGLE takes as input a database of examples and produces both a set of human-readable decision rules and a program (in FORTRAN, Pascal, BASIC, or C) that expresses the same rules. It's $398 for the IBM PC or $1,198 for the VAX. Reader Service No. 17.
PML Systems
3139 East Almond Ave.
Orange, CA 92669
(714) 771-7744

A new product line from **Arity Corp.** supports software development in PROLOG on the IBM PC. The line includes five products: the PROLOG Compiler and Interpreter for $795, the Interpreter alone for $350, the Expert Systems Development Package for $295, the SQL Development Package for $295, the Arity Screen Design Toolkit for $49.95, and the Arity File Interchange Toolkit for $49.95. The Combination Pack, which contains all five products, costs $1,225. Reader Service No. 18.
Arity Corp.
358 Baker Ave.
Concord, MA 01742
(617) 371-1243

For the Macintosh

The **WSM Group** has released Hyper-C 68000 for the Macintosh computer. Hyper-C is a full K & R C compiler with extensions to allow a natural interface with the Macintosh ROM toolbox routines. Mac toolbox calls are generated in-line, so there's no need for any special interfacing code. Pascal functions can be coded in-line, and assembly source code is produced as output. Full SANE (Standard Apple Numeric Environment) support is provided. Hyper-C is priced at $199.95 and has no licensing requirements. Reader Service No. 19.
The WSM Group Inc.
1161 N. El Dorado Pl.
Ste. 241
Tucson, AZ 85715
(602) 298-7910

Pascal Extender and C Extender from **Invention Software Corp.** are compiled libraries designed to simplify the task of programming the Macintosh interface and reduce development time. They support all standard toolbox commands plus add high- and medium-level routines for creating and manipulating windows, menus, controls, scrollbars, dialogs, and alert boxes. The Extender handles graphics and text scrolling and window activation. It fully supports text editing, desk accessories, and graphics printing. Pascal Extender retails for $69.95; C Extender retails for $129.95. Reader Service No. 20.
Invention Software Corp.
P.O. Box 3168
Ann Arbor, MI 48106
(313) 996-8108

For Atari ST

The Pro Pascal language compiler from **Prospero Software** is available on the Atari ST. It includes strings, 7- and 16-digit precision floating point, separate compilation, and 4-byte integers. Pro Pascal has full GEM AES and VDI bindings. It costs $149. Reader Service No. 21.
Prospero Software Ltd.
190 Castelnau
London SW13 9DH
England
011 441 741-8531

Let's Write is a new word processor for the Atari ST from **Mark Williams Co.** It features advanced text processing, a spelling checker, and communications in one package. The editor gives the user up to 11 windows to view and change text from multiple files. The text formatter is similar to Unix's nroff. Let's Write costs $79.95. Reader Service No. 22.
Mark Williams Co.
1430 W. Wrightwood Ave.
Chicago, IL 60614
(312) 472-6659

Hardware for the PC

American Computer & Peripheral has introduced the American Abovefunction Card, a multifunction memory board for American IBM PC/XTs and compatibles with full support of Lotus, Intel, and Microsoft expanded-memory features. Supporting up to 2 megabytes of expanded memory, the Abovefunction Card also provides serial, parallel, and game ports and a real-time clock/calendar. It includes a utility disk that contains Expanded Memory Manager, RAM disk, print buffer, real-time clock/calendar program, and example CONFIG.SYS and AUTOEXEC.BAT files. The card has a suggested list price of $380 with no RAM or $820 with 2 megabytes RAM installed. Reader Service No. 23.
American Computer & Peripheral Inc.
2720 Croddy Way
Santa Ana, CA 92704
(714) 545-2004

DigiBoard has announced DigiRam/3MB, a memory-expansion board for the IBM PC/AT that provides up to 3 megabytes of error-checked RAM on a single board. The board has split memory addressing, filling up to 640K and continuing

OF INTEREST
(continued from page 118)

at 1 megabyte and up. It is user-upgradable and available in any desired memory configuration. Reader Service No. 24.
DigiBoard Inc.
6751 Oxford St.
St. Louis Park, MN 55426
(612) 922-8055

Everex Systems has introduced an EGA video card that is compatible with the IBM Enhanced Graphics Adapter, includes a parallel printer port and 256K display memory on-board, and is supplied with the company's proprietary EGMODE software. The Enhancer board provides 640×350-resolution graphics in 16 colors from a palette of 64 colors. It has a suggested retail price of $425. Reader Service No. 25.
Everex Systems Inc.
48431 Milmont Dr.
Fremont, CA 94539
(415) 498-1111

Pacific Data Products is offering the V68K line of intelligent graphics boards. These high-resolution graphics cards are full-size, IBM PC add-on cards that feature palette sizes of up to 16 million colors, a Motorola MC68010 video processor, and a capacity of up to 2 megabytes of video memory. Palettes can be reloaded every scan line with no screen performance degradation. The boards are compatible with the IBM PC, PC/XT, PC/AT, and RT/PC. The MC68010 video processor can perform vector to raster conversion or other tasks. Reader Service No. 26.
Pacific Data Products
8545 Arjons Dr., Ste. 1
San Diego, CA 92126
(619) 549-0136

Univation has announced a new multifunction accelerator card called the Dream Board for the IBM PC. It combines up to 2 megabytes RAM with a 200–400 percent increase in speed, serial and/or parallel ports, a clock/calendar, an optional 8087 math chip, and several utilities to speed disk I/O. The card retails for $595–$795 with 512K RAM, depending on options selected. Reader Service No. 27.
Univation
1231 California Cir.
Milpitas, CA 95025
(408) 263-1200

Earth Computers' 287-Power-10 is a 10-MHz math coprocessor board for the IBM PC/AT that plugs into the existing 80286 socket. The product line also includes 5- and 8-MHz versions of the board. Prices range from $249 for the 5-MHz version to $695 for the 10-MHz version. Reader Service No. 28.
Earth Computers
P.O. Box 8067
Fountain Valley, CA 92728
(714) 964-5784

Networking

ITT has introduced the Xtra XL, a high-performance supermicrocomputer running both DOS and Xenix. The system, based on the Intel 80286 processor, is designed to maximize the computing power available to users operating in local-area network and shared-processor environments and maintains existing industry standards. Xtra XL includes 8-MHz, zero-wait-state memory; dynamic disk I/O caching; an average hard-disk access time of 28 milliseconds; and an optional 80287 math coprocessor. In addition, an 8-MHz 80186-based communications coprocessor offers dramatic throughput speed in multiuser configurations. Models I and II are intended for use as local-area-network servers operating under ITT DOS 3.1. Model I, priced at $5,299, includes 640K RAM, a 1.2-megabyte floppy disk, and a 40-megabyte hard disk. Model II, priced at $7,299, includes 640K RAM, a 1.2-megabyte floppy disk, and a 72-megabyte hard disk. Reader Service No. 29.
ITT Information Systems
2350 Qume Dr.
San Jose, CA 95131
(408) 945-8950

A local-area-network configuration for the AT&T 6300 series of microcomputers has been announced by **The Destek Group.** The new configuration uses industry-standard CSMA/CD media-access protocols with a network bus speed of 2

OF INTEREST
(continued from page 120)

megabits/second. It features increased buffer memory and circuitry. Prices vary according to configuration. Reader Service No. 30.
The Destek Group
830 E. Evelyn Ave.
Sunnyvale, CA 94086
(408) 737-7211

Network-OS is a local-area-network operating system from **CBIS** designed to allow users of IBM PCs, PC/XTs, PC/ATs, and compatibles to create microcomputer-based LANs. It is fully NetBIOS/DOS 3.1 compatible and can support all major LAN topologies including token ring. Network-OS also supports Novell file and record locking and can run virtually any third-party DOS 3.1 application. The retail list price of Network-OS is $995 per 16 users. Reader Service No. 31.
CBIS Inc.
2323 Cheshire Bridge Rd.
Atlanta, GA 30324
(404) 634-3079

A simultaneous voice/data multiplexer for use on four-wire voice grade lines is available from **Coherent Communications Systems Corp.** The SVD-2400 overlays a full-duplex, 2,400-bps data channel to an existing leased line, allowing it to support both voice and data traffic. Reader Service No. 32.
Coherent Communications Systems Corp.
60 Commerce Dr.
Hauppauge, NY 11788
(516) 231-1550

A high-speed, 2,400-bps, stand-alone modem designed for personal computers and terminals is available from **Prentice Corp.** The P-224 is a full-duplex modem that meets CCITT V.22 bus recommendations and Bell 212A and 103 standards and supports the Hayes AT command set. It features auto-answer and auto-dial operation and can be used with touch-tone or pulse-dial phones. Standard features also include automatic bit rate and parity selection and auto-speed recognition on answer. Reader Service No. 33.
Prentice Corp.
266 Caspian Dr.
Sunnyvale, CA 94088-3544
(408) 734-9810

InterContinental Micro Systems has released TurboDOS/PC, a package that runs on an IBM PC, a compatible, or any 8086-line microcomputer system that uses MS-DOS or PC-DOS, Versions 1.x, 2.x, or 3.0. TurboDOS/PC allows the PC to become a TurboDOS network client and to access the disk drives and printers belonging to the TurboDOS file and print servers in the network. The single-copy list price for TurboDOS/PC is $100. Reader Service No. 34.
InterContinental Micro Systems
4015 Leaverton Ct.
Anaheim, CA 92807
(714) 630-0964

Woolf Software Systems has announced Move-It, Version 4, a communications package for microcomputer users. The new version has automatic file compression, keyboard macros, scripting files, XMODEM protocol support, infilter and outfilter commands, and the ability to send and receive files automatically. Move-It, Version 4, retails for $150. Reader Service No. 35.
Woolf Software Systems
6754 Eton Ave.
Canoga Park, CA 91303
(818) 703-8112

A serial communications board that allows users to interconnect up to six IBM PCs, PC/XTs, PC/ATs, and compatibles using the EasyLAN local-area network is available from **Server Technology.** The Com Port Board-6 is based on the RS-232 interface standard and works in conjunction with the DOS-supported COM1 and COM2 ports. By incorporating a Server Com Port Board-6 along with the DOS-supported COM1 and COM2, users can interconnect a total of eight PCs using EasyLAN. If desired, up to six PCs can be interconnected to the Com Port Board-6, with the COM1 and COM2 reserved for other serial devices. Three boards can be accommodated per PC, allowing up to 18 PCs to be interconected in an EasyLAN network. An EasyLAN starter kit is priced at $179.95. Reader Service No. 36.
Server Technology Inc.
1095 E. Duane, Ste. 107
Sunnyvale, CA 94086
(408) 738-8377

Languages

Clarion from **Barrington Systems** is a structured programming language designed for commercial applications. Clarion runs on IBM PCs or compatibles, requires a hard-disk drive and a minimum of 320K. The utility programs are integrated; a single keystroke can terminate one utility, then load and execute the next. Screen and report layouts are designed interactively. The entire package sells for $295. Reader Service No. 37.
Barrington Systems Inc.
150 E. Sample Rd.
Pompano Beach, FL 33064
(305) 785-4555

A new set of Forth example programs, the Forth Model Library (volumes 1–3), is now available from the **Forth Interest Group.** The library includes application programs compatible with Forth-83 systems available from the most popular Forth vendors. The volumes are *A Forth List Handler* by Martin J. Tracy, *A Forth Spreadsheet* by Craig A. Lindley, and *Automatic Structure Charts* by Kim R. Harris. Each volume is available for $40. Reader Service No. 38.
Forth Interest Group
P.O. Box 8231
San Jose, CA 95155
(408) 277-0668

Smalltalk-AT from **Softsmarts** includes the Xerox Smalltalk-80 source code, the Xerox image, and Softsmarts' ST-80 virtual machine. With Smalltalk-AT, users can run any application developed on a larger dedicated Smalltalk machine. It requres an IBM PC/AT with 640K base memory, at least 512K expansion memory, a Mouse Systems' three-button mouse, and the IBM Enhanced Graphics Adapter. The total package price is $995. Reader Service No. 39.
Softsmarts Inc.
4 Skyline Dr.
Woodside, CA 94062
(415) 327-8100

Software Express has released Version 1.6 of Apgen, a fourth-generation language in the Unix marketplace. The new version features a set of training tutorials. It is compatible with all previous versions and sells for $6,000. Reader Service No. 40.
Software Express
2925 Briarpark Dr.,
7th Floor
Houston, TX 77042
(713) 974-2298

Star Value Software has announced four software development tools for Modula-2 programmers: Textio, a display and printer I/O library; Graphix, an in-

OF INTEREST

(continued from page 123)

terface to the MetaWindow system from MetaGraphics; and Make and XRef, utilities for managing large-size development projects. These tools are designed to work in conjunction with the Logitech Modula-2/86 development system on IBM PCs or compatibles. All four products are sold separately and include complete documentation. Textio and Graphix are $79 each; Make and XRef are $59 each. Reader Service No. 40.
Star Value Software
12218 Scribe Dr.
Austin, TX 78759
(512) 837-5498

Wordcraft's C: A Programming Workshop teaches the C programming language interactively. The workshop includes an integrated editor and standard compiler. The test module reports whether a program exercise gives correct results. Users can also develop C functions with no disk delay. The Workshop runs on IBM PCs and compatibles with 19K. It costs $39.95 plus shipping and handling. Reader Service No. 41.
Wordcraft
3827 Penniman Ave.
Oakland, CA 94619
(415) 534-2212

Version 1.5 of Mystic Pascal from **Mystic Canyon Software** features screen output, a complete on-line help library, and fast execution speed. It has a full-screen editor, multitasking operating system, ISO Pascal compiler, and interactive debugger mode. Users' programs can occupy up to 640K of storage for code and data, and users can run up to 100 program sections concurrently. Reader Service No. 42.
Mystic Canyon Software
P.O. Box 1010
Pecos, NM 87552
(505) 757-6344

Graham Software Corp. has introduced Version 1.3 of Alice: The Personal Pascal. This version is compatible with industry-standard Pascal compilers and supports Borland International's Turbo Pascal. Alice: The Personal Pascal consists of four, integrated, computer language products: an IBM PC-compatible Pascal interpreter, a language intelligent editor, on-line help facilities, and a full-function debugging system. Version 1.3 has a suggested retail price of $95 (U.S.) or $129 (Canada). Reader Service No. 43.
Graham Software Corp.
4 Kingwood Pl.
Kingwood, TX 77339
(713) 359-1024

QuickSilver Software has released memory-resident reference guides for popular compilers. These reference guides provide clear documentation on the procedural and syntactical elements of each language. The packages require 128K RAM, one disk drive, and PC-DOS 2.0 or later. Each guide costs $16.95. Reader Service No. 44.
QuickSilver Software
P.O. Box 880887
San Diego, CA 92108
(619) 543-9896

DDJ

ADVERTISER INDEX

*This advertiser prefers to be contacted directly: see ad for phone number.

ADVERTISING SALES OFFICES

Midwest
Michele Beaty **(317) 875-8093**

Southeast
Gary George **(404) 355-4128**

Northern California/Northwest
Lisa Boudreau **(415) 366-3600**

Southern California/AZ/NM/TX
Michael Wiener **(415) 366-3600**

Advertising Director
Robert Horton **(415) 366-3600**

SAS Institute Inc. Announces

Lattice C Compilers for Your IBM Mainframe

Two years ago...

SAS Institute launched an effort to develop a subset of the SAS® Software System for the IBM Personal Computer. After careful study, we agreed that C was the programming language of choice. And that the Lattice® C compiler offered the quality, speed, and efficiency we needed.

One year ago...

Development had progressed so well that we expanded our efforts to include the entire SAS System on a PC, written in C. And to insure that the language, syntax, and commands would be identical across all operating systems, we decided that all future versions of the SAS System—regardless of hardware—would be derived from the same source code written in C. That meant that we needed a C compiler for IBM 370 mainframes. And it had to be good, since all our software products would depend on it.

So we approached Lattice, Inc. and asked if we could implement a version of the Lattice C compiler for IBM mainframes. With Lattice, Inc.'s agreement, development began and progressed rapidly.

Today...

Our efforts are complete—we have a first-rate IBM 370 C compiler. And we are pleased to offer this development tool to you. Now you can write in a single language that is source code compatible with your IBM mainframe and your IBM PC. We have faithfully implemented not only the language, but also the supporting library and environment.

Features of the Lattice C compiler for the 370 include:

- **Generation of reentrant object code.** Reentrancy allows many users to share the same code. Reentrancy is not an easy feature to achieve on the 370, especially if you use non-constant external variables, but we did it.
- **Optimization of the generated code.** We know the 370 instruction set and the various 370 operating environments. We have over 100 staff years of assembler language systems experience on our development team.
- **Generated code executable in both 24-bit and 31-bit addressing modes.** You can run compiled programs above the 16 megabyte line in MVS/XA.
- **Generated code identical for OS and CMS operating systems.** You can move modules between MVS and CMS without even recompiling.
- **Complete libraries.** We have implemented all the library routines described by Kernighan and Ritchie (the informal C standard), and all the library routines supported by Lattice (except operating system dependent routines), plus extensions for dealing with 370 operating environments directly. Especially significant is our byte-addressable Unix®-style I/O access method.
- **Built-in functions.** Many of the traditional string handling functions are available as built-in functions, generating in-line machine code rather than function calls. Your call to move a string can result in just one MVC instruction rather than a function call and a loop.

In addition to mainframe software development, you can also use our new cross-compiler to develop PC software on your IBM mainframe. With our cross-compiler, you can compile Lattice C programs on your mainframe and generate object code ready to download to your PC.

With the cross-compiler, we also offer PLINK86™ and PLIB86™ by Phoenix Software Associates Ltd. The Phoenix link-editor and library management facility can bind several compiled programs on the mainframe and download immediately executable modules to your PC.

Tomorrow...

We believe that the C language offers the SAS System the path to true portability and maintainability. And we believe that other companies will make similar strategic decisions about C. Already, C is taught in most college computer science curriculums, and is replacing older languages in many. And almost every computer introduced to the market now has a C compiler.

C, the language of choice...

C supports structured programming with superior control features for conditionals, iteration, and case selection. C is good for data structures, with its elegant implementation of structures and pointers. C is conducive to portable coding. It is simple to adjust for the size differences of data elements on different machines.

Continuous support...

At SAS Institute, we support all our products. You license them annually; we support them continuously. You get updates at no additional charge. We have a continuing commitment to make our compiler better and better. We have the ultimate incentive—all our software products depend on it.

For more information...

Complete and mail the coupon today. Because we've got the development tool for your tomorrow.

SAS®

SAS Institute Inc.
SAS Circle, Box 8000
Cary, NC 27511-8000
Telephone (919) 467-8000 x 7000

I want to learn more about:

☐ the C compiler for MVS software developers
☐ the C compiler for CMS software developers
☐ the cross-compiler with PLINK86 and PLIB86

today...so I'll be ready for tomorrow.

Please complete or attach your business card.

Name ____________________

Title ____________________

Company ____________________

Address ____________________

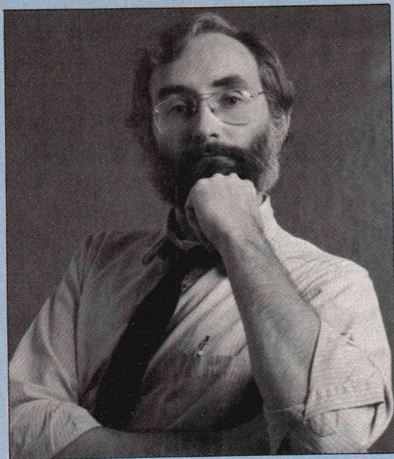
City ____________________ State ________ ZIP ________

Telephone ____________________

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA. 27511-8000. Telephone (919) 467-8000, x 7000

DDJ 9/86

SWAINE'S FLAMES



I've noticed that all the best columnists make lists—lists of their favorite products, lists of announced but unreleased products, lists of their favorite announced but unreleased products. I think I ought to make some lists.

A list can be a sequence, and a sequence can be a puzzle, as in, What is the next item in this sequence?

```
1.  00010100    00010110
    00010001    00000111
    00010110    00000001
    00001111

2.  STOA  MACRO   PAR
          IFC     EQ,*PAR*A*
          SA6     PAR
          ENDIF
          ENDM

3.  while remainder < > 0 do
      begin
          m := n;
          n := remainder;
          remainder := m mod n
      end;

4.  PLOT PIE
    HEADING 'PAGES OF SOURCE CODE'
    SHOW SLICE FOR 'DDJ' EXPLODED
    COMBINE SLICES LT 5 PERCENT
    END
```

Pencils down. Anyone who said . . .

```
5.  omnipotent(X) :- can(X,_,_).
    indestructible(X) :-
         not(can(_,destroy,X)).
    omnipotent(god).
    ?-can(god,create,X),
         indestructible(X).
```

gets to stay after class to clean erasers.

The sequence was, of course, first- through nth-generation language code. There is a clear progression, I think, in information density per statement as you go up the generations. The progression in readability is also clear, from the cryptic bit-stream of first-generation binary to the English-like code of fourth-generation RAMIS II.

When you get to the PROLOG code, though, the readability progression crumbles. PROLOG is about as readable as Pascal, and there are other reasons to question PROLOG's position in the sequence.

Given appropriate definitions of *create* and *destroy*, the PROLOG program above will answer the question, Can an omnipotent being create an indestructible object? You can actually go to your computer, load PROLOG, key in this program, and get an answer. Resolution of this classic antinomy would be an event in the history of logic, but I suspect that my four lines of code say more about PROLOG's failures as a logical language than about gods or logic. Clocksin and Mellish also seem to acknowledge that PROLOG may not be the language of fifth-generation programming by calling it "a potential basis for an important new generation of programming languages. . . ."

I really don't think we're there yet.

My cousin Corbett has been telling me lately that *DDJ* needs a new large software project. *DDJ*, he reminds me, gave the first micro programmers a language that would fit into their tiny memory spaces with Tiny BASIC in 1976. Four years later, the magazine published the first version of Small-C, again shoehorning a language into the limited memory space of micros.

But now microcomputers have megabytes of memory and validated Ada implementations. Does this *DDJ* kind of minimization make sense any more? Corbett thinks so. Paradoxically, he holds that the key is to think big about thinking small. That's the principle behind his new software development project, for which he wants to solicit programmers through the pages of *DDJ*. There is, Corbett, maintains, only one possible next element in the sequence that began with Tiny BASIC and Small-C: Tiny Star Wars. Send to the usual address for your starter disk and security clearance.

But I don't seem to be getting the hang of this list thing. The most common kind of columnist list may be the classic bitch list. The tone can range from Andy Rooney-whiney to Ian Shoales-snarly. Here's my snarl.

I'm tired of waiting for Alan Kay's Dynabook, Ted Nelson's Xanadu and Hypertext, Adam Osborne's software pricing watershed, the home market, telecommunications software that doesn't push all the tough decisions off on me, hardware design that doesn't penalize me for being left-handed. All the best programmers are left-handed. I was tired of "near-letter quality"; now I'm tired of "near-typeset quality." When will printers be good enough that their manufacturers don't have to apologize for their output? I'm tired of waiting for Americans to recognize the value of a dollar and of a vote, to stop spending both on trash, and to demand competence from industry and government. Either that or demand printer-style labeling: "near-Sony quality," "near-Gorbachev lucidity." And I'm tired of the lingering death of copy protection. Pull the plug on that baby.

I'm tired. I gotta go.

Michael Swaine

Michael Swaine
editor-in-chief

Test File Name	HOT C (V.2.33)	Lattice (V.2.15)	Microsoft (V.3.00)
fillscr()	37 sec	61 sec	50 sec
scroll()	45 sec	62 sec	56 sec
prtf()	45 sec	65 sec	58 sec
cpyblk()	62 sec	167 sec	56 sec
cpychr()	38 sec	87 sec	49 sec
diskio()	38 sec	103 sec	41 sec
array()	52 sec	69 sec	77 sec
optimize()	35 sec	32 sec	60 sec
sieve()	93 sec	106 sec	96 sec
rsieve()	54 sec	107 sec	54 sec
minmain()	452 bytes	10260 bytes	1928 bytes
minprtf()	2786 bytes	11684 bytes	5750 bytes
minputs()	1384 bytes	10228 bytes	4296 bytes
minfio()	2694 bytes	10794 bytes	5896 bytes
Price:	$99	$500	$500

HOT C really is.

It's fast. It's powerful. It writes tighter, faster, more efficient code.

And it's cheap. So now just $99 (suggested retail price) gets you cutting edge performance without the ragged edge problems.

HOT C is an optimizing single-pass compiler. It includes an assembler, linker, debugger, librarian and library source code. It supports large- and small-memory models and all operators and types except bit fields, with outstanding error and type checking. And in performance, it meets or beats (sometimes dramatically) every other compiler we've come up against.

HOT C is a great power trip for novices, too. A comprehensive tutorial, an impressive text editor and totally revised documentation will have you speaking C like a native in no time flat.

So try HOT C today. Now you can have the steak *and* the sizzle.

For details, just drop a note to WordTech Systems, Inc., P.O. Box 1747, Orinda, CA 94563.

Or call our HOTline at (415) 254-0900.

HOT C
by WORDTECH

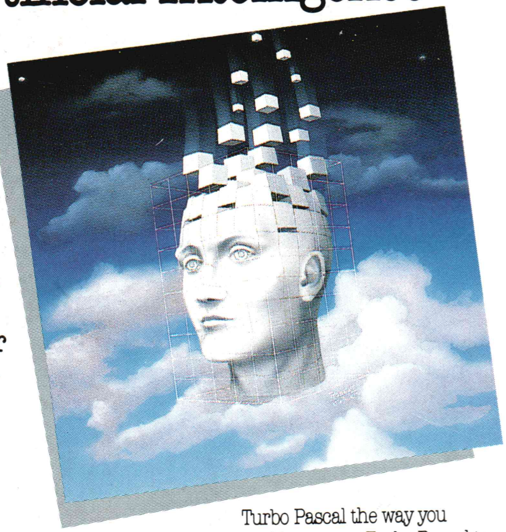
Circle **no. 208** on reader service card.

Step-by-step tutorial, demo programs with source code included!

Borland's new Turbo Prolog is the powerful, completely natural introduction to Artificial Intelligence

Prolog is probably one of the most powerful computer programming languages ever conceived, which is why we've made it our *second* language—and "turbocharged" it to create Turbo Prolog.™

Our new Turbo Prolog, the natural language of **A**rtificial **I**ntelligence, brings supercomputer power to your IBM® PC and introduces you step-by-step to the fascinating new world of **A**rtificial **I**ntelligence. And does all this for an astounding $99.95.



Turbo Prolog is to Prolog what Turbo Pascal® is to Pascal!

Our Turbo Pascal astonished everyone who thought of Pascal as "just another language." We changed all that—and now Turbo Pascal is the de facto worldwide standard, with hundreds of thousands of enthusiasts and users in universities, research centers, schools, and with professional programmers, students, and hobbyists.

You can expect at least the same impact from Turbo Prolog, because while Turbo Prolog is the most revolutionary and natural programming language, it is also a complete development environment—just like Turbo Pascal.

> "Turbo Prolog offers generally the fastest and most approachable implementation of Prolog.
>
> Darryl Rubin, AI Expert"

Even if you've never programmed before, our free tutorial will get you started right away

You'll get started right away because we have included a complete step-by-step tutorial as part of the 200-page Turbo Prolog Reference Manual. Our tutorial will take you by the hand and teach you everything you're likely to need to know about Turbo Prolog and artificial intelligence.

For example: once you've completed the tutorial, you'll be able to design your own expert systems utilizing Turbo Prolog's powerful problem-solving capabilities.

Think of Turbo Prolog as a high-speed electronic detective. First you feed it information and teach it rules. Then Turbo Prolog "thinks" the problem through and comes up with all the reasonable answers—almost instantly.

If you think that this is amazing, you just need to remember that Turbo Prolog is a 5th-generation language—and the kind of language that 21st century computers will use routinely. In fact, you can compare Turbo Prolog to Turbo Pascal the way you could compare Turbo Pascal to machine language.

You get the complete Turbo Prolog programming system for only $99.95

You get a complete Turbo Prolog development system including:

- The lightning-fast Turbo Prolog incremental compiler and the interactive Turbo Prolog editor.
- The 200-page reference manual which includes the step-by-step Turbo Prolog tutorial.
- The *free* GeoBase™ natural query language database including commented source code on disk—ready to compile. GeoBase is a complete database designed and developed around U.S. geography. It includes cities, mountains, rivers, and highways, and comes complete with natural query language. Use GeoBase immediately "as is," or modify it to fit your own interests.

So don't delay—don't waste a second—get Turbo Prolog now. $99.95 is an amazingly small price to pay to become an immediate authority—an instant expert on artificial intelligence! The 21st century is only one phone call away.

Turbo Prolog 1.0 Technical Specifications

Programming System Features

- ☑ *Compiler:* Incremental compiler generating native in-line code and linkable object modules. The linking format includes a linker and is compatible with the PC-DOS linker. Large memory model support. Compiles over 2500 lines per minute on a standard IBM PC.
- ☑ *Interactive Editor:* The system includes a powerful interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code. At run-time, Turbo Prolog programs can call the editor, and view the running program's source code.
- ☑ *Type System:* A flexible object-oriented type system is supported.
- ☑ *Windowing Support:* The system supports both graphic and text windows.
- ☑ *Input/Output:* Full I/O facilities, including formatted I/O, streams, and random access files.
- ☑ *Numeric Ranges:* Integers: −32767 to 32767; Reals: 1E−307 to 1E+308
- ☑ *Debugging:* Complete built-in trace debugging capabilities allowing single stepping of programs.

YES! I want the best

Turbo Prolog at only:

$99.95

To order by phone, or for a dealer nearest you,

Call (800) 255-8008

in CA call (800) 742-1133.

Send me ___ Turbo Prolog at $ _____

Outside USA add $10 per copy

CA and MA res. add applicable sales tax $ _____

Amount enclosed: $ _____

This price includes shipping to all US cities

Payment: VISA MC Bank Draft Check

Credit card expiration date: ___/___

Card # | | | | | | | | | | | | |

†You must have an IBM or true compatible running DOS 2.0 or later. SU11

My computer's name and model is: _____

The disk size I use is: ☐ 3½" ☐ 5¼"

NOT COPY PROTECTED

***60-DAY MONEY-BACK GUARANTEE**

Name: _____

Shipping Address: _____

City: _____

State: _____ Zip: _____

Telephone: _____

CODs and purchase orders WILL NOT be accepted by Borland. Outside USA make payment by bank draft payable in US dollars drawn on a US bank.

*YES, if within 60 days of purchase this product does not perform in accordance with our claims, please call our customer service department and we will gladly arrange a refund.

†Minimum system requirements:
IBM PC, XT, AT, PCjr, and true compatibles; 384K RAM.

BORLAND
INTERNATIONAL

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

Vive la différence

Circle **no. 161** on reader service card.